Raisonance Tools for ARM

Raisonance Tools for ARM core-based microcontrollers

Getting started

Document version
28 September 2011

Contents

1. Introduction

This guide should be used by anyone with an interest in Ride7 for ARM.
The new Ride7 and RKit-ARM **must** be registered to operate correctly.

1.1 Purpose of this manual

This manual helps you get started using any of the Raisonance products that are offered by Raisonance's application development solution for ARM core-based MCUs. It describes how to compile and debug your application or one of the included sample applications. It assumes that you have the prerequisite knowledge of the C and ARM / thumb / thumb2 assembly languages and CPUs.

1.2 Scope of this manual

The tools specifically addressed in this manual include:

- GCC C compiler toolchain
- Ride7 development environment
- RFlasher7 programming interface
- RLink debugger programmer

1.3 Related documents

Each tool in this document has its own user manual. These documents are provided with the Raisonance software installation and in the support section of the Raisonance internet site www.mcu-raisonance.com.

- RLink user manual
- Using the Open4 (EvoPrimer) user manual
- REva v3 user manual
- REva v2 and earlier user manual
- REva STM32 daughter boards user manual
- REva STRx daughter boards user manual
- Ride7 Online help (provided with installation)
- RFlasher7 user manual
- GCC toolchain user documentation (provided with installation)

1.4 Additional help or information

Please visit the Raisonance website: *http://www.raisonance.com/* and the forum *http://www.raisonance.com/Forum/punbb/* or contact Raisonance.

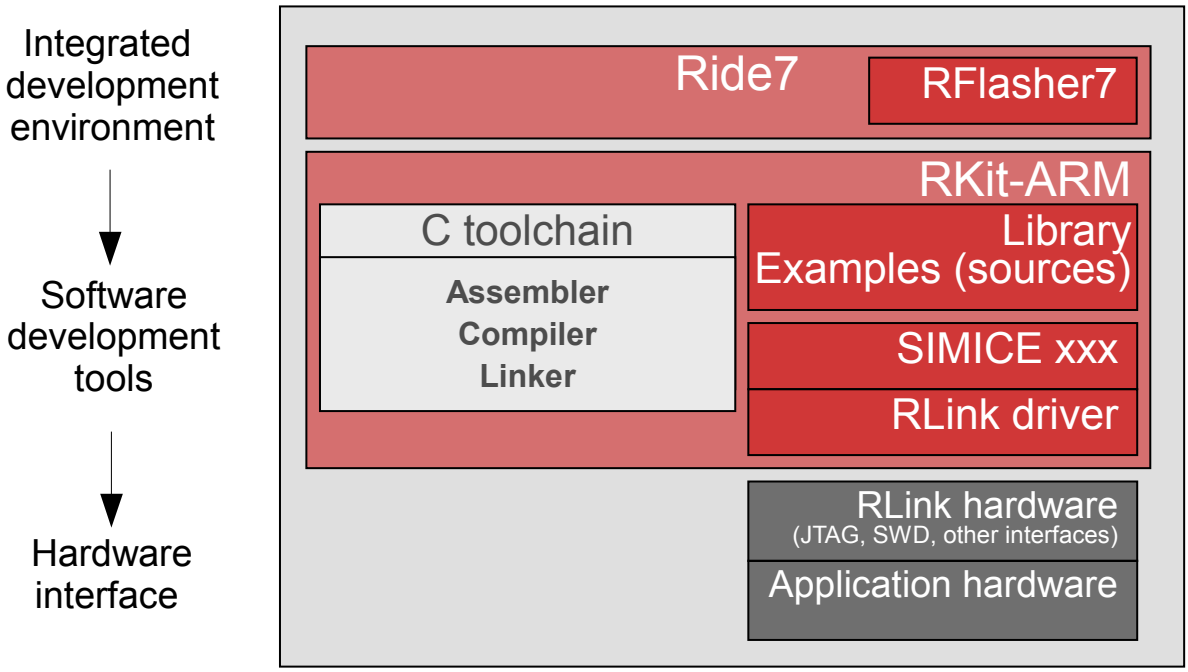
Address: Raisonance S.A.S.
17, Avenue Jean Kuntzmann,
38330 Montbonnot Saint Martin
France
Telephone: +33 4 76 61 02 30
Fax: +33 4 76 41 81 68
Email: ***support@raisonance.com***

If you find any errors or omissions, or if you have suggestions for improving this manual, please let us know by email.

2. Raisonance tools for ARM overview

Raisonance's integrated development environment, Ride7 for ARM®, interfaces with a range of development tools including:

- Editor, debugger and project manager that is common to several ST microcontroller families of ARM cores and which integrates the GNU GCC toolchain.
- RKit-ARM.
- Software-based simulator.
- RLink USB to JTAG dongle from Raisonance.
- JTAGjet USB to JTAG dongle from Signum Systems (for STRx devices only).
- Debugging interface to use with the Raisonance SIMICE ARM simulator, or to connect to your ARM CPU for programming and debugging with JTAG standard emulators.



2.1 Ride7 and RFlasher7

Ride7 acts as the user interface for all the other tools. It gives you start-to-finish control of application development including code editing, compilation, optimization and debugging.

The associated RFlasher7 interface can program the Flash memory of the target MCU.

2.2 RKit-ARM

The RKit-ARM includes a C toolchain (GCC) controlled directly from Ride7, and defines the devices, script tools, and debugging and programming interface features (see Section 7 for details).

- **Lite** license includes default Raisonance hardware tools, GCC toolchain, and limited software features.
- **Enterprise** license includes support for 3rd party compilers and C++ programming.

2.3 SIMICE ARM simulator

Raisonance's SIMICE ARM simulator can simulate some of the ARM peripherals, or connect to your ARM CPU for programming and debugging with JTAG standard emulators such as:

- RLink USB to JTAG dongle from Raisonance,
- JTAGjet USB to JTAG dongle from Signum Systems (for STRx devices only).

SIMICE ARM is included in RKit-ARM-Lite. It simulates the core (including the entire memory space) and most peripherals. Complex peripherals (USB, CAN) and some less common peripherals are not simulated.

The same user interface is used for the simulator and the hardware debugging tools (RLink, JTAGjet).

2.4 RLink

RLink is a JTAG standard emulator with a USB interface. It allows you to program ARM devices on an application board and debug the application while it runs on the target. Raisonance's REva evaluation boards feature an embedded (detachable) RLink.

Rlink-STD for ARM (and Primer, REva, Open4) are limited to a code size of 64 Kbytes.

RLink-PRO for ARM allows debugging with unlimited code size.

Note: RLinks have different capabilities for programming and debugging ARM7, ARM9, Cortex-M3, ST7 and uPSD microcontrollers. Your RLink's capability to program and debug any Ride7-supported target microcontroller is shown when Ride7 reads your RLink's serial number.
For a description of the different debugging capabilities, refer to *Debugging with Hardware Tools*.

2.5 Licenses

Any files not mentioned here are under license by Raisonance or other companies. They should not be copied or distributed without written agreement from their owners.

- The GNU toolchain is under the GPL license, which makes it free to use without royalties, as are some of the files written by Raisonance and ST. You can assume that everything under the *arm-gcc* and *GNU* subdirectories of the Ride7 installation folder can be freely used, copied and distributed, but is without any warranty and only limited support from Raisonance.
- Raisonance's solution for ARM core based MCUs has two levels of license:
  - Hardware license: controls the hardware-based debug limitation of 64 Kbytes of code in RAM or Flash memory.
  - Software license: controls software features, details are provided at http://www.mcu-raisonance.com/software_packages_arm.html

2.6 Supported devices and tools

2.6.1 ARM MCUs

The ARM-core based microcontrollers addressed by this solution include:

- ARM Cortex (M3, M4, M0)
  - STMicroelectronics (STM32F, STM32L)
  - NXP (LPC17xx, LPC11xx)
  - Texas Instruments (Stellaris)

Support is provided for some sub-families that are based on legacy ARM9 and ARM7 cores, including:

- ARM966E (supported legacy devices) STMicroelectronics (STR9)
- ARM7TDMI (supported legacy devices)
  - STMicroelectronics (STR7)
  - NXP (LPC21xx, LPC23xx, LPC24xx)

For a complete listing of specific supported MCUs and derivatives refer to the list of selectable MCUs in the Ride7 project settings. For this purpose, users can install Ride7 with an evaluation version of RKit-ARM (Download at: http://www.mcu-raisonance.com/arm-download.html). Newly supported derivatives are reported in the release notes for each version.

2.6.2 Derivatives

Ride7 for ARM supports most of the existing STR7x, STR9x and STM32 derivatives to various degrees. It also supports some ARM7 NXP LPCxx devices, and Cortex M3 LPC1768 devices.

The up-to-date list of supported derivatives and the limitations to the software simulation can be seen in the **Target Options** in Ride7. A history of new supported devices can be seen in the release notes.

2.6.3 Third party tools that can be used in conjunction with Ride7 for ARM

Ride7 for ARM can be used together with a number of third-party tools including:

- **GNU GCC toolchain** for ARM® (ARM-none-eabi-gcc, ARM-none-eabi-as, ARM-none-eabi-ld): they allow you to compile applications in assembler and/or C language. Ride7 automatically installs and calls the free open-source GNU toolchain. See http://www.gnu.org/ for more information about GNU programs.
- The **Phyton CodeMaster-ARM** toolchain (with the Enterprise version only). This can be purchased as a bundle with RKit-ARM (contact sales@raisonance.com for pricing information). The CodeMaster-ARM software can be used directly from the Ride7 interface, although it should be installed separately.
- **JTAGjet**: this JTAG standard emulator with USB interface from Signum Systems allows you to program the STRx on an application board and debug the application while it runs on the STRx. It uses the JTAG protocol. For more information on using the JTAGjet, refer to: *Debugging with Hardware Tools*.
  JTAGjet-ETM provides powerful real-time trace capabilities, for STR9 **only**.

3. How to register the new Raisonance tools for ARM

The new Ride7 and RKit-ARM must be registered to operate correctly. Registration requires a Raisonance hardware product or software product license that is under a valid support contract (a standard support contract expires one year after the date of purchase).

Products that allow activation of RKit-ARM include the RKit-ARM license (Serial number or Dongle) and RLink (RLink-STD, RLink-PRO, REva starter kits, Primers).

Unregistered software functions for a 7-day evaluation period with full features of the Enterprise version. After 7 days the software can no longer be used. If this occurs, contact info@raisonance.com.

3.1 Install the new software

Perform these steps to install your new software:

1. Remove old versions of Ride7 and RKits.
2. Install the new Ride7 software, then the RKit-ARM software, then validate its operation (test compilation, connection to RLink and to target CPU, etc.).
3. Launch **Ride7** and select **Help > License**. Activate your software by providing:
   a. Software serial number (purchasers of RKit-xxx-Enterprise or RKit-xxx-Lite tool sets).
   b. Hardware dongle (purchasers of RKit-xxx-Enterprise or RKit-xxx-Lite tool sets).
   c. RLink serial number (purchasers of RLink, REva, or a Raisonance Primer).
4. Register the software by selecting the menu item **Help > License**...

Note: During the registration process, you will be notified if your product's support contract has expired. This notification includes information about how to renew your tool's support contract

3.2 Register using a serial key

You must use this procedure for:

- Old RLink, REva, or Primers (serial number prior to dngXXXXXX18000).
- RLink-STD, REva or Primers purchased more than one year ago, that have been upgraded to "Pro".
- RKit-ARM-Enterprise software licenses (Serial Number or Dongle).

If your product was purchased:

- less than one year ago, you will need to provide your proof of purchase.
- more than one year ago, you will need to purchase the (annual) support extension.

Perform these steps to register using a serial key:

1. Contact Raisonance (info@raisonance.com) to obtain a **Serial Key**. They need your Software or RLink Serial Number (For RLink, click **Connect to RLink** in RFlasher7 or Ride7 **debug options**).
2. Log on as admin, ensure you have internet access, a working default browser and a working email address.
3. Open Ride7 or RFlasher7. If it does not open automatically, click **Help->About Ride7**.
4. Click the **Change License** button. Select the **Serial Activation** method, click **Next**.
5. Enter the **Serial Key** provided to you. Click **Next**.
6. Connect to the internet and click **Get Activation code online** to open your browser on the Raisonance server.
7. Fill in the form and click **Generate and Send Activation code**.

8. Check your server for an email from Raisonance Support team with the **Activation Code**.
9. Copy the **Activation Code** from the email to the Ride7 window. Click **Finish**.
10. Close Ride7 and open it again. Check in the **About Ride7** window that you are in "Lite Suite" for RKit-ARM and "Standard version" for Ride7. Otherwise please contact our technical support (support@raisonance.com).

3.3 Register using a dongle

1. Plug in the dongle and make sure the LED lights up (otherwise you must install the driver).
2. Open Ride7 or RFlasher7. If it does not open automatically click **Help->About Ride7**.
3. Click the **Change License** button. Select **Dongle Activation**, click **Next**.
4. Close Ride7 and open it again. Check in the **About Ride7** window that you are in "Lite Suite" for RKit-ARM and "Standard version" for Ride7, otherwise contact our technical support (support@raisonance.com).

3.4 Register using an RLink

Automatic registration can be performed using an RLink-STD, RLink-PRO, REva starter kit or Primer.

Automatic registration requires that you have the PC connected to the internet and a hardware serial number, or software serial key that is covered by a valid support contract.

Follow these steps to register your RLink automatically:

1. Log on as admin, ensure you have internet access, a working default browser and a working email address.
2. Plug in the RLink and make sure the BUSY LED turns OFF (USB enumeration).
3. Open Ride7 or RFlasher7. If it does not open automatically click **Help->About Ride7**.
4. Click the **Change License** button. Select **RLink Activation**, click **Next**.
5. If the software finds the RLink, it says **RLink detected. Ready for registration**. Click **Next**.
6. Connect to the internet and click **Get Activation code online** to open your browser on the Raisonance server.
7. Fill in the form and click **Generate and Send Activation code**.
8. Check your server for an email from Raisonance Support team with the **Activation Code**.
9. Copy the **Activation Code** from the email to the Ride7 window. Click **Finish**.
10. Close Ride7 and open it again. Check in the **About Ride7** window that you are in "Lite Suite" for RKit-ARM and "Standard version" for Ride7. Otherwise please contact our technical support (support@raisonance.com).

4. Creating a project

Assembly and C applications can be written using the free GNU GCC toolchain. This chapter gives an overview of how to create an ARM project with Ride7.

Notes for users of other GCC toolchains: You may have to modify the GCC_EXEC_PREFIX environment variable to rectify compatibility issues between different GCC toolchains. If you have this kind of problem, look at the GCC documentation to see the usage of this variable.

4.1 Example project

Ride7 for ARM provides several example projects that are ready to run. One of them is very simple and you should look at it first. To open this project, use **Project > Open Project** in Ride7 and select the project *testR7.rprj*. For standard installations of Ride7, this file is found in *<Ride>\Examples\ARM\Test*. The example is described in source file comments. Other examples are located in the same directory.

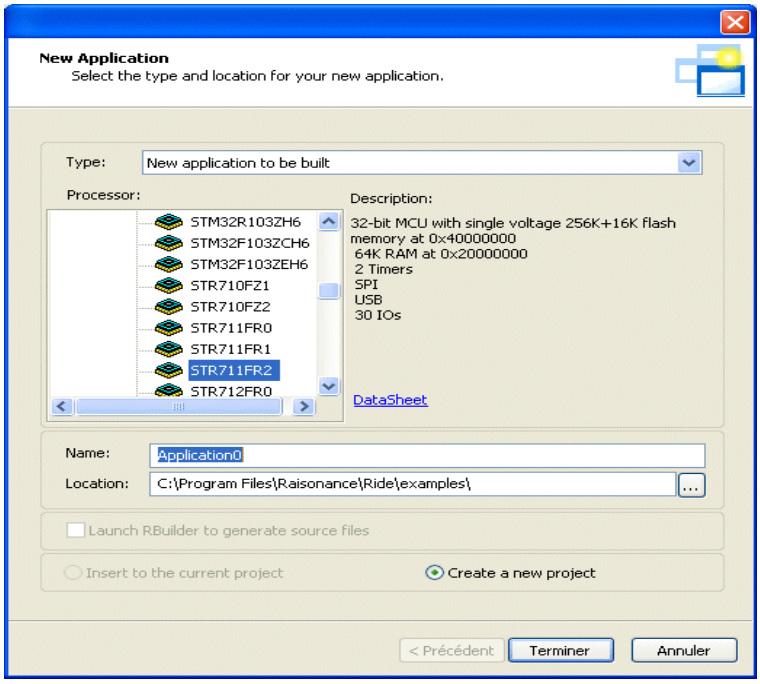
4.2 Creating a new project

All RKit users can use the GNU tools to create a project, as explained in section 4.2.1.

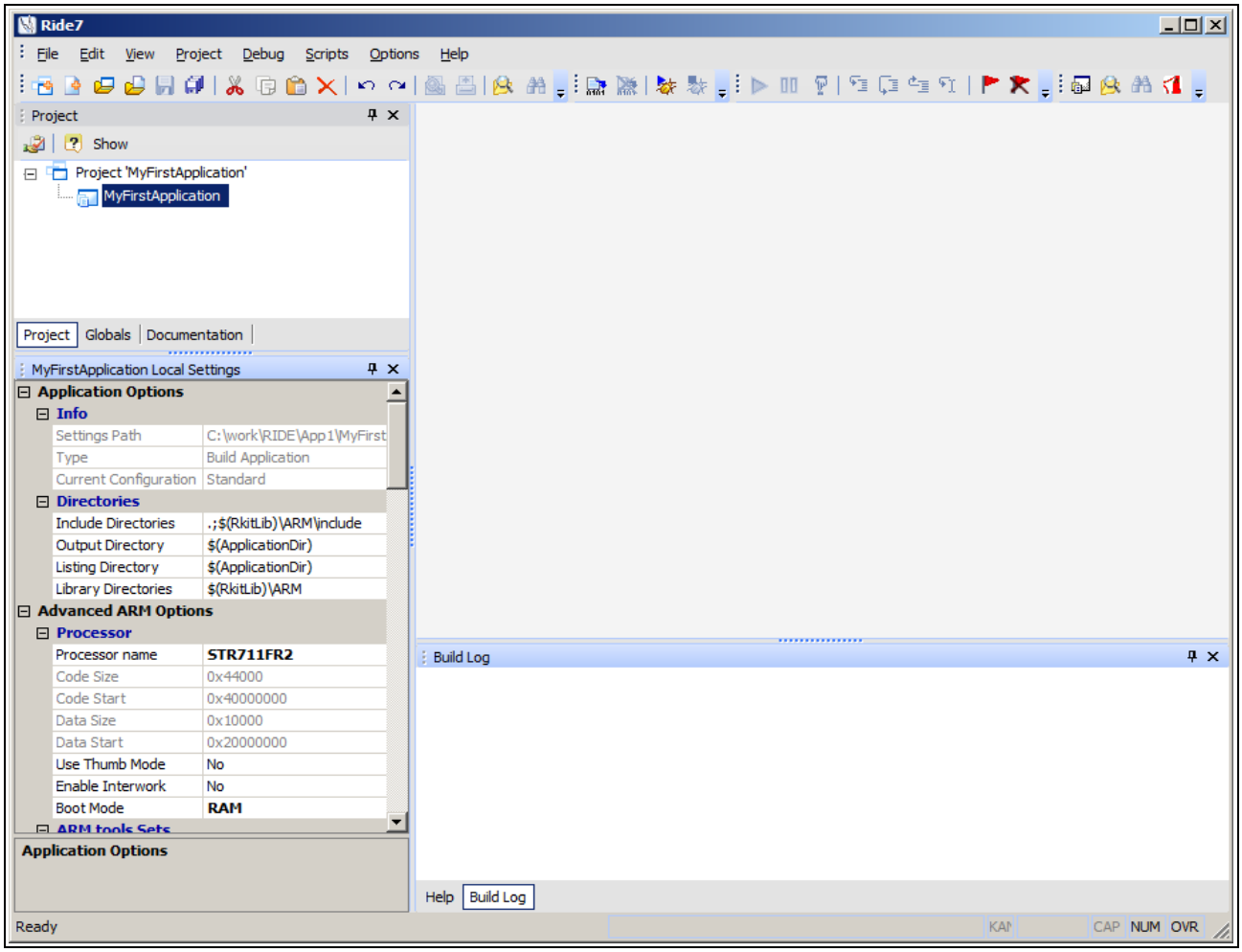
Enterprise versions can use the Generic compiler interface, as explained in section 4.2.2.

4.2.1 Using the GNU tools

1. In Ride7, go to the menu **Project > New Project**.
2. Select **New application to be built**, then your target **processor**.
3. Choose the application name and the path to the main application folder.
4. Click on **Finish** to generate your application. Your application project is now created.



The Ride7 environment should look like this:



- **Boot Mode** option (visible in the **Advanced ARM Options**) determines the memory space containing the image of the code that the target executes after a reset.
  You can select **FLASH**, **RAM**, or **External Memory**. For information about boot modes, refer to your device datasheet and the section *Boot mode choices*. For your first try, choose **FLASH**.
- **Use Thumb Mode** option allows you to choose between the Thumb or ARM modes, for ARM7 and ARM9 devices. See the device datasheet for more information. Thumb2 mode is automatically selected when using a Cortex-M3 device.
- **Enable Interwork** option allows you to select a different ARM/Thumb mode for each source file of the project (this option is not valid on Cortex microcontrollers).

You are now ready to add files to the project, build and debug it as explained in the Ride7 manual (common to all targets). Click **Help > View Documentation** to see it.

You can also change the GNU configuration, to use the Ride7 libraries for example, as explained in section 4.4.

4.2.2 Using Phyton CMC-ARM Compiler Kit (Enterprise version)

To select a different third party compiler tool, in the **Application Options**, select **ARM toolset > Build toolset > Phyton tools**.

The Compiler, Assembler and Linker sections are all used in a similar way: Enter a build command line for each of the tools.

Each tool command line should be composed of the following elements:

- It must start with the tool name with its full (system-dependent) path. This tool path name should be enclosed in double-quotes.
  For instance the linker command-line may start with the following tool name:

```
"C:\Program Files\Phyton CMC-ARM Compiler Kit\Bin\MCLINK.EXE"
```

- It can then contain additional command-line arguments, such as:

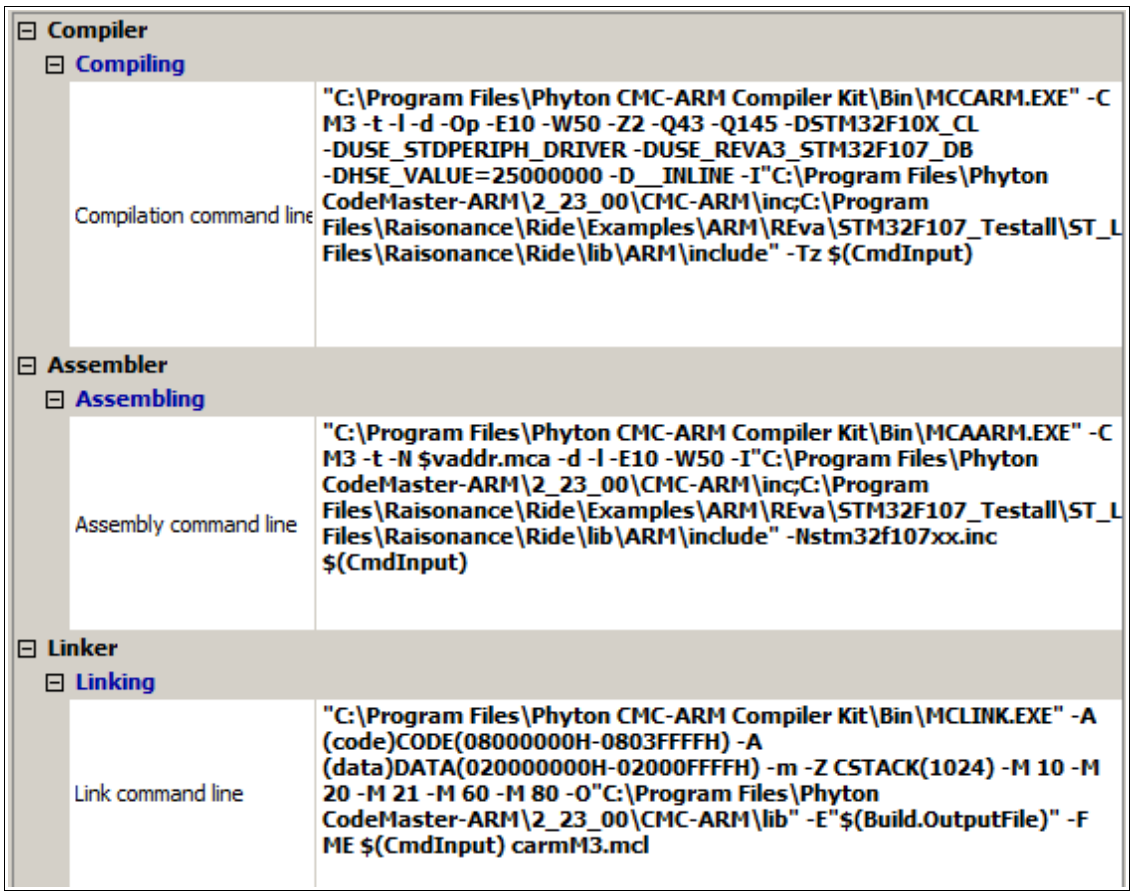
```
-A (code)CODE(08000000H-0803FFFFH) -A (data)DATA(020000000H-02000FFFFH) -m -Z
CSTACK(1024) -M 10 -M 20 -M 21 -M 60 -M 80
```

- It can contain prebuild macros that will be replaced by Ride7 at the moment the tool is executed. For instance, the following Ride7 macros can be used:

```
$(CmdInput)           file(s) that will be used as input to the command
$(Build.OutputFile)   full path and name of the output file
```

In the Ride7 project manager, you can then right click on a Source file and select **Run compiler/ Run assembler/Run linker**, or select the project and **Build**.

Here is an example of Phyton CMC-ARM Compiler Kit commands:



4.2.2.1 Startup code

When using the Phyton CMC-ARM Compiler Kit, you must provide your own application startup file. The automatic startup builder is available only for the GNU tools.

Example startup files are provided with the Phyton CMC-ARM Compiler Kit in order to help you in this process.

4.2.2.2 Example application

You can find a Phyton CMC-ARM Compiler Kit example for Ride7 in:

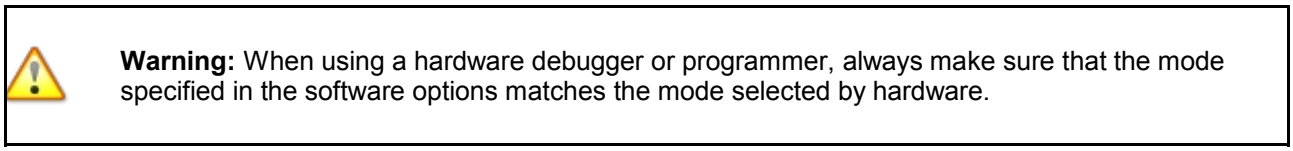
```
%RideDir%examples\ARM\REva\STM32F107_Testall\TestAll_Phyton.rprj
```

4.3 Boot mode choices

After a reset, the ARM microcontrollers start executing the code at address zero:

- For the STR71x, STR75x and STM32x, this code is an image of one of the other memory spaces – Flash, RAM, or External Memory. The boot mode is determined by the state of specific pins at reset. See your device datasheet for more information.
- For the STR73x, STR91x, and LPCxx, only the Flash can be mapped at 0. However, a pseudo RAM mode can be managed by Ride7: the application is loaded in RAM and the reset vector just jumps to the RAM segment.

Flash is the standard mode, which your application will very probably use in the final product. You may prefer to use RAM mode if you require more breakpoints for debugging, as long as the RAM is large enough to hold the program. RAM mode cannot be used in the final application because RAM content is modified by a power down.



Warning: When using a hardware debugger or programmer, always make sure that the mode specified in the software options matches the mode selected by hardware.

4.3.1 Flash mode

For all devices startup code is placed at the start of Flash by the linker. The rest of the code is placed after the startup in Flash.

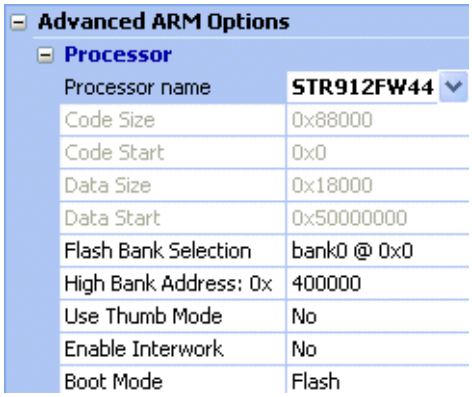
4.3.1.1 STR7 and STM32 devices

Flash is also seen at address 0. The data initialization values are also placed in Flash, and then copied to RAM by the startup code. The read-only data is also placed in the Flash and is directly accessed there during the execution of the application.

4.3.1.2 STR9 devices

By default, bank0 of Flash resides at address 0 (e.g. CSX = 0) and bank1 (32KB) resides at address 0x400000.
However, you can modify the option **Flash Bank Selection**:

- By selecting **bank1 @ 0x0**, you invert the respective locations of banks 0 and 1 (bank0 will reside at the address 0x400000),
- By modifying the **High Bank Address** value, you force the relocation address of the upper bank (either bank1 or bank0). This address must be larger than the size of the bank at address 0, and must be a multiple of the size of the other bank.



The data initialization values are placed in Flash, and copied to RAM by the startup code.

The read-only data is also placed in Flash and is directly accessed from here during the execution of the application.

4.3.2 RAM mode (debug mode)

You should use this mode during the development phase of your project for faster hardware debugging (it can also be used with the software simulator, but then it offers no advantage in terms of download/programming time and number of breakpoints).

In RAM mode:

- For the STR71x, the RAM is at 0x20000000 but it is also seen at 0x00000000.
- For the STR73x, the RAM is physically at 0xA0000000 but it is also seen at 0x00000000.

- For the STR75x, the RAM is physically at 0x40000000 but it is also seen at 0x00000000.
- For the STR91x, the RAM is physically at 0x04000000. The reset and interrupt vectors are placed in the Flash and jump to some RAM addresses.
- For the STM32, the RAM is physically at 0x20000000.
- For the LPCx, the RAM is physically at 0x40000000. The reset and interrupt vectors are placed in the Flash and jump to some RAM addresses.

In this mode, the linker places the code and data segments in the RAM and the data initialization values in the Flash.

The final application cannot use this mode because the RAM is volatile and has to be reloaded at every power-up of the microcontroller. It can be used during debugging because Ride7 is able to load the RAM at the beginning of every debug session. The constants and the initialization values for the global variables are still stored in the Flash. Therefore, do not be surprised if Ride7 has to erase and program the Flash when starting a debug session in RAM mode.

4.3.3 External memory mode

Use of this mode depends on the microcontroller and is for experienced users only, you must consult the relevant datasheet for details. You can still use Ride7 as a project manager and software simulator, but you cannot debug or program. It uses external memory for booting, the target external memory space is seen at address 0. RLinks and JTAGjet do not work in this mode.

4.4 GNU GCC toolchain configuration

Ride7 Options Manager (**Options > Project Properties**) provides the most important options needed to configure the GCC toolchain. The GNU options are separated into 3 sections:

- GCC Compiler
- AS Assembler
- LD Linker

Only the sections that apply to the selected project node and its children are displayed, for example:

- If a C source file is selected, only GCC Compiler section is visible
- If the application or project node is selected, all three sections are visible

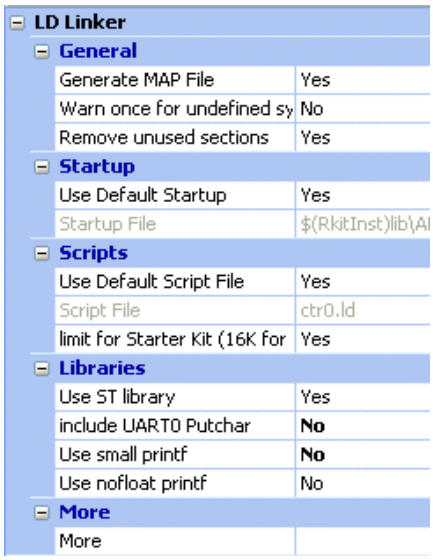
Note: When you modify options on a child node (most probably a source file), you create a superset of local options for this node and its children. If you want to globally modify an option (this is the case most of the time), do not forget to verify that the application node is selected, and not a child node.

4.4.1 Compiler and Assembler options

Refer to the tools specific documentation for a detailed description of the GCC Compiler and Assembler options.

As Ride7 provides additional libraries this is explained in more detail here.

4.4.2 LD linker options



The LD Linker provides various options:

- General
- Startup
- Scripts
- Libraries. Ride7 provides some libraries which you can choose to include (or not) by (un)selecting the corresponding options.
- More
- C++ applications (RKit-ARM Enterprise version only).

When you are using the Thumb instruction set, or Interworked mode, the libraries used are the corresponding Thumb libraries. Thumb libraries have the same name as the non-thumb libraries, with *_thumb* appended.

4.4.2.1 General

Generate MAP file: Makes the Linker produce a map file (.MAP).

Warn once for undefined symbols: If checked, only one warning is displayed for each undefined symbol, rather than once per module which refers to it.

Remove unused sections: If checked, the linker does not link the sections that are not used. Activate this together with the GCC Compiler option **Per function sections** in order to have the linker remove any unused function from the application.

4.4.2.2 Startup

Default startup: Set to **Yes** to use the default startup file. Set to **No** to use your own startup file.

Startup File: Indicate the path of your own startup file (or not if your startup file is part of the source files). You can see the default startup and linker script files provided by Ride7 in the directory *<Ride>\lib* (which you can copy and modify).

Use GCC Startups: The default option **No** adds the '-nostartfiles' switch to the linker command line. That removes the standard GCC startups, which are not usually required in embedded C applications and might interfere with the startup provided by Ride7 or the chip manufacturer (or at least adds useless code). However, these files must be included for C++ applications because they hold some constructors that are required.

4.4.2.3 Scripts

Use Default script file: Set to **Yes** to use the default script file. Set to **No** to use your own script file, and fill in the Script File box (you must use a linker script).

Script file: Indicate the path of the linker script file that you want to use.

- The linker script (in several parts) is generated just before the link. If you want to see the script used for your application, just link your project and look in the application's directory for the associated script file. For example, if your application is called *MyApplication.elf*, the script generated is called *MyApplication.elf.ld.*
  This primary script is generated at each link, and includes the input files, output files, and main linker script, which is either the default or the custom linker script that you specified using the **Script File** option.
  The main default script includes three sub-scripts.
- The virtual device STRx-TEST has no default linker script, therefore you must use a custom linker script when you use this device.

Starter Kit Limited: This option should be used when debugging in RAM with the limited version of the RLink, such as the RLinks embedded in the REva evaluation boards, and the standard RLinks.

- Because these RLinks only access the lower 64Kbytes of each memory area (RAM, Flash, etc.), the linker must initialize the stack pointer at an address lower than 64K and not at the very end of the RAM as it would do otherwise.
- Also, it generates a link error if your application uses more than this reduced amount of RAM. If you are using the simulator, an RLink with "Pro" capabilities, JTAGJet, or if you are only programming the ARM's Flash (without debugging it), then you should not select this option because it greatly reduces the possibilities of the system.

4.4.2.4 Libraries

Use ST Precompiled library (STRx):

- This option only appears when the selected device is an STRx device.
- When set (by default), this option adds a precompiled library to the application, containing a standard API designed by STMicroelectronics for handling the target peripherals.
- Library files are available for every STRx (in thumb and ARM modes) target supported. The library sources can be found in this folder: *<Ride>\LIB\ARM\<family>_LIB* but you should download the latest version from ST's website. You can use and distribute them freely.

Note: This option should not be used for C++ applications, because the libraries are compiled for C only. To use these libraries in a C++ application, you must include the library sources in your project. See the section about C++ applications for more information.

Use OLD Precompiled library (STM32x)

- This option only appears when the selected device is a STM32x device.
- When set (NOT set by default), this option adds a precompiled library to the application containing a standard API designed by STMicroelectronics for handling the target peripherals.
- THIS OPTION IS DEPRECATED.
  It should only be used for compiling old projects, created using old versions of the RKit-ARM.
  It cannot be used in C++ applications.
  The normal way to use the ST library is to include its sources in your application's project.
  We recommend using the latest versions from ST's website, the package there includes Ride7 example projects showing you how to use them, by including the libraries sources in the project.
  (The STM32 examples provided with Ride7 also show how this should be done, but the source may be obsolete.)

Note: Other manufacturers (NXP, TI) also provide peripheral libraries for their CPUs. You should find them on these manufacturer's websites.

Include UART0 putchar

- This option only appears for some target devices.
- This option adds *io_putchar.a* (or *io_putchar_thumb.a*), which includes standard `putchar` and `getchar` functions using the UART0.
- It should be used in conjunction with *UART0_stdio.h* (instead of *stdio.h*).
- It redirects the output for functions using `stdout`, like `printf`, to the UART0.
- The "TEST" example shows how to use it.
- The library source can be found in *<Ride>\LIB\ARM\IO_PUTCHAR\…*
- You can use and distribute them freely.

printf capabilities

- This option adds a library which includes a reduced version of `printf`.
- The standard version from l*ibc.a* ("Full GNU printf") includes many rarely used things..

- The reduced version ("Small printf") links the *<Ride>\lib\ARM\small_printf(_thumb).a* library, which includes a complete `printf`, reduced for embedded applications.
- The no-float version ("no-float printf") links the *<Ride>\lib\ARM\e_stdio(_thumb).a* library, which includes a even more reduced `printf` that does not handle floating points.
- These libraries (small and no-float) have been written by Raisonance but are free and open-source. The sources of the libraries can be found in
  <Ride>\lib\ARM\small_printf\…
  <Ride>\lib\ARM\e_stdio\…
  You can use and distribute them freely.

These libraries call the `__io_putchar` function to send characters to their destination. (see "`UART0 putchar`" section above)

You just need to provide your own `__io_putchar` function if you want to redirect the output of `printf` to whichever output channel(s) you wish. (UART1, etc.)

C++ Libraries

- This option tells the linker to include the C++ libraries in addition to the C libraries, and is required for linking C++ applications. See the section about C++ for more information.
- This option MUST NOT be activated when building C applications.

4.4.2.5 More

This option lets you specify options that will be added to the command line. Those options are added just before the linker configuration file in the command line. Look at the LD documentation for more information on the available commands and switches.

4.4.2.6 Creating a C++ project (Enterprise)

Although the GCC toolchain allows building of C++ applications, Ride7 is designed to provide the best experience to C users, not C++. Even so, the Enterprise version of the RKit-ARM can be used to build C++ applications.

To create a C++ application, you need to proceed as described above for creating a C application, with the following changes:

- Change the **C++ Libraries** linker option to **Yes**. You can only do that if you are using the Enterprise version of the RKit-ARM.
- If you are using the default linker script, make sure you set the **Starter Kit Limited** option (described above) to **No**. This is required because even the simplest C++ application is larger than the 64K limitation.
- Use a custom startup file (see above) and make sure it calls the `__libc_init_array` function for executing static constructors before calling `main`. Most default startup codes, from Raisonance, ST or others, DO NOT include this call because they are optimized for C applications. So you need to add it yourself for C++ applications.
- Include the GCC startup files by setting the **Use GCC Startups** linker option to '**Yes**'. These startups will include the C++ constructors.
- Include at least one C++ file in your project. Ride7 and GCC use the file extension to recognize that it is C or C++. All C++ files should have the *cpp* extension.

After that, building, programming and debugging the application works exactly as for a C application.

You will find an example of Ride7 project for a C++ application here:

<Ride>\Examples\ARM\REva\STM32F103_toggle_cpp

It implements all the points listed above. It is an interesting example of these points even if you work on other CPUs.

Note: Debugging C++ applications with the the RKit-ARM Lite version is not supported. It may work in some specific situations, however it is not included in the support agreement.

5. Debugging with the simulator

The Ride7 for ARM simulator simulates ARM7, ARM9 and Cortex-M3 cores and most common peripherals, it lets you check your code and the interaction between your code and the peripherals. This section shows use of the simulator for ARM and Cortex-M3 microcontrollers. The interface is the same for all targets. Detailed documentation is in the Ride7 general documentation.

5.1 About the simulator

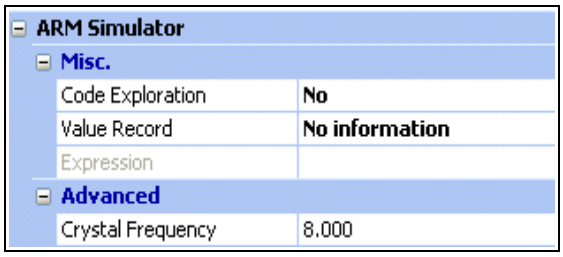
Ride7 supports most STRx and STM32 derivatives to various degrees, and some LCPx devices.

In this section, we use the example project **testR7**, found in the Ride7 installation directory: *<Ride>\Examples\ARM\Test\TestR7.rprj.* This project uses the *General Purpose Input/Output Port 1* peripheral and consists of a new empty project with one very simple *main.c* file.

5.2 Simulator options

Misc.

- **Code Exploration**: If set to **Yes**, code is explored from the load, to recognize and to flag the first byte of any instruction (this is a call-tree exploration). As instructions have different sizes, this is needed to display an exact disassembly to the user. This option must be checked in order for the trace to explore the disassembly code.
  If set to **No**, loading is faster and exploration is not complete; the code is displayed as *db 0xxh.* In simulation, you should explore progressively (`explore` command).



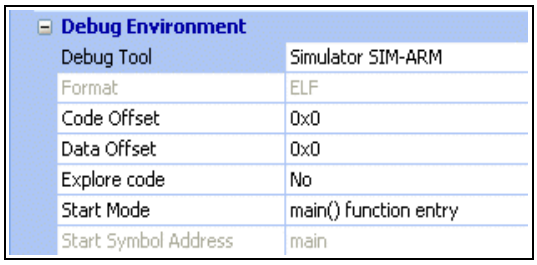
- **Value Record**: This option allows to record, or not, additional information at each instruction. SP indicates the value of the stack pointer after the execution of the current instruction
- **Expression**: Enter a simple expression to be evaluated.

Advanced: **Crystal Frequency**: Set the crystal frequency you want to simulate.

5.3 Launching the simulator

To launch the simulator: type **CTRL–D** or select **Debug > Start** in the main menu.

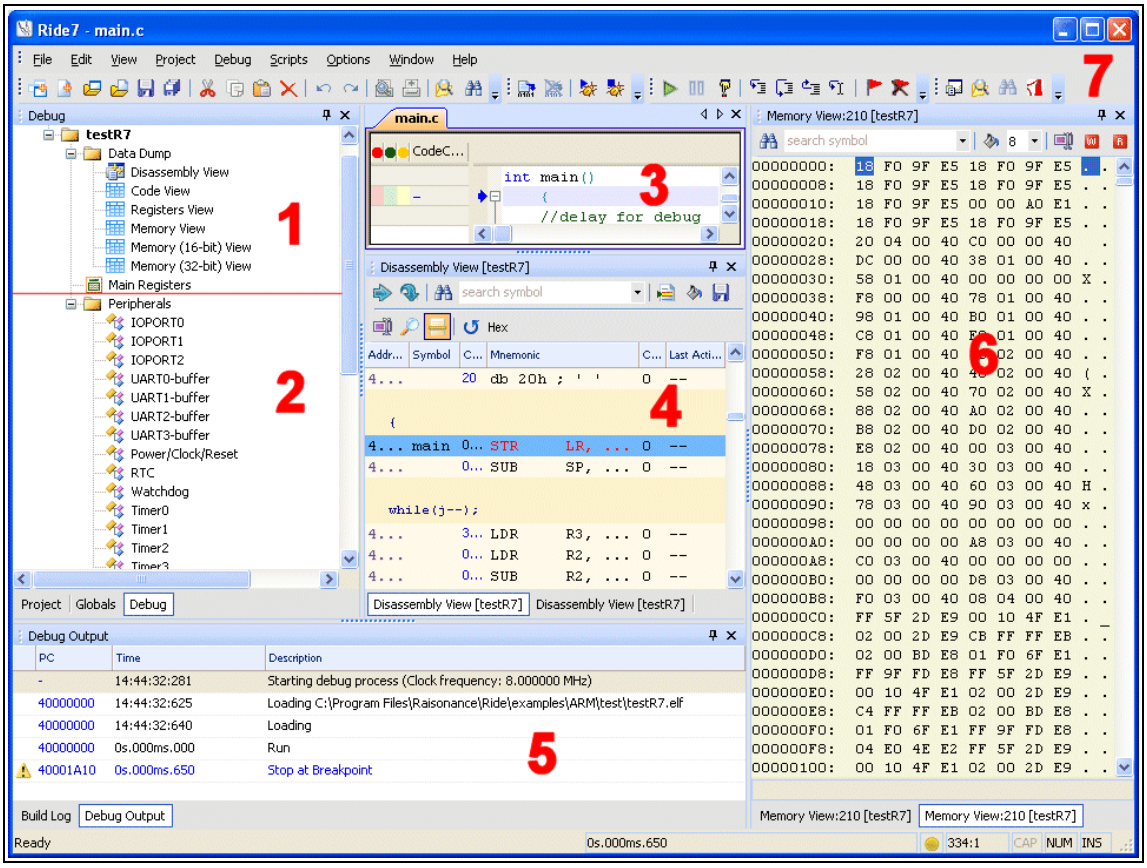
Before launching the simulator, check the **Debug Environment** options, in particular that **Simulator SIM-ARM** is selected for **Debug Tool**.



If your project has not been built, this is done automatically before launching the simulator, otherwise the simulator is launched directly.

You are now in the simulator.

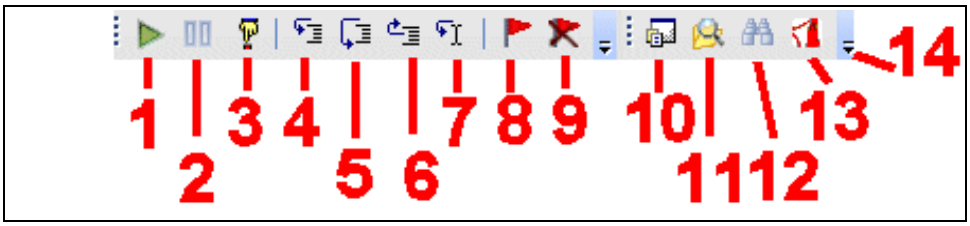
Your Ride7 window looks like the following image:



1. The upper part of the **Debug** tab. Shows the different Data views available on a given microcontroller. To see a specific view, double-click on its name.
2. The lower part of the **Debug** tab. Shows the peripherals available on a given microcontroller. To see a specific peripheral, double-click on its name. Most peripherals are NOT simulated.
3. The **source file** as edited in C or in assembly language. The green circles on the left indicate lines that contain instructions where you can place breakpoints. The line highlighted in blue indicates the current PC. This is the next instruction to be executed.
4. The **Disassembly View.** Displays the instruction to be executed by the simulator. It is a dump of the memory where the code is located. The blue arrow at the beginning of the line indicates the current PC, as in the source window. The following columns are available:
   - **Address:** address where the instruction is located.
   - **Symbol:** name of the symbol, if a symbol is located at this address.
   - **Code:** byte-code located at this address.
   - **Mnemonic:** mnemonic corresponding to the byte-code.
   - **Code Coverage:** number of times byte-code at this address has been executed.
   - **Last action:** most significant effect of the instruction during its last execution.
5. The **Debug Output** window provides feedback on the status of debugging process. Status information can include errors and debug event logs. Some message lines have hyperlinks to the Disassembly view at the PC address where the event occurred.
6. **Data Dumps Views** (here **Memory View**) is only available during a debug session, and shows the content of the different memory dumps. The addresses associated with symbols are highlighted in pink. A status bar on the bottom of this view displays the symbols and read/write accesses. You can modify the content of the dump by selecting the value you want to alter, typing the new value and pressing **Enter,**.or double click the ASCII value and type the value in ASCII format directly.
7. The toolbar, which allows you to control the simulation (see the next section for information).

5.4 Simulator toolbar

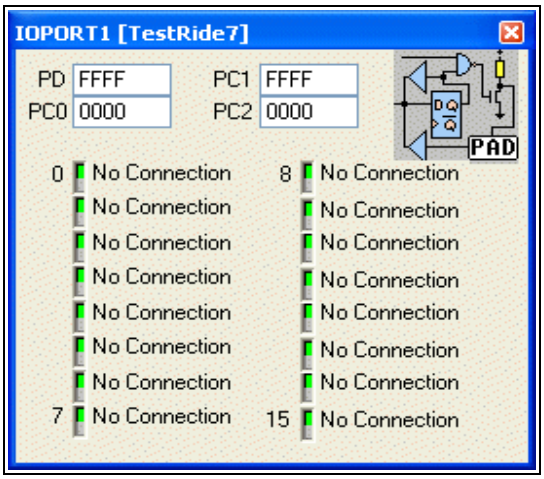
The simulation is controlled by the simulator **toolbar**:



1. **Run**: Pressing this button launches the application. When the application is running, this button is grayed, then the Pause button becomes available.
2. **Pause**: Pressing this button stops the application.
3. **Reset**: Pressing this button resets the application.
4. **Step Into**: On a function call in a line of the C source code, this button steps into the called function. If it is not a function call, it goes to the next line in the source code.
5. **Step Over**: On a function call in a line of the C source code, this button steps over the called function.
6. **Step Out**: In a function this button gets out of the function.
7. **Run To**: Run the program until the cursor.
8. **Toggle breakpoint**: If there is no breakpoint on the current line, Ride7 sets a breakpoint on it. If there is one, the breakpoint is removed.
9. **Clear All breakpoints**: Clear all the breakpoints set.
10. **Project**: Open the **Project** window.
11. **Find in files**: Pressing this button opens the **Find in files** window allowing the search an expression in files of the project directory.
12. **Binoculars:** This opens the search window.
13. **Documentation:** Pressing this button opens the **Documentation** window allowing to open *Help html* files.
14. **Toolbar Options:** Allows to **Add** or **Remove** buttons in the menu.

5.5 Viewing a peripheral

To view a peripheral, you must open it by clicking on the corresponding item in the peripheral tree. For example, to view the Port 1, double click on the **IOPORT1** icon. The Port 1 view appears.



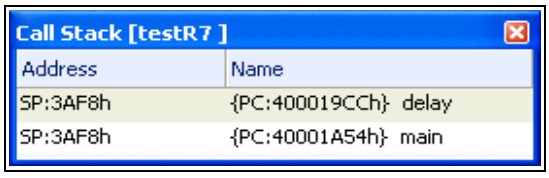
This view shows the state of each of the port's pins. Green indicates a value of one, and red a value of zero. It is possible to connect each pin of the port to a Net, to VCC, to Ground or no connection. This is done by clicking on the **LED**. The registers also let you control the peripheral.

With the test application described above, click on the **Run** button  to launch the execution and click on the **Pause** button  . You then see the LEDs counting.

Note: Currently, only UART is fully simulated in order to provide *putchar/printf* capabilities.
For the other peripherals, the views provide a comprehensive presentation of the internal registers, but the dynamic behavior of these registers exists only when running the program on real hardware via a JTAG connection (see *Debugging with Hardware Tools*).

5.6 Viewing the stack

You can view the stack by clicking **View >Debug Windows > View Call Stack**. This opens this window:

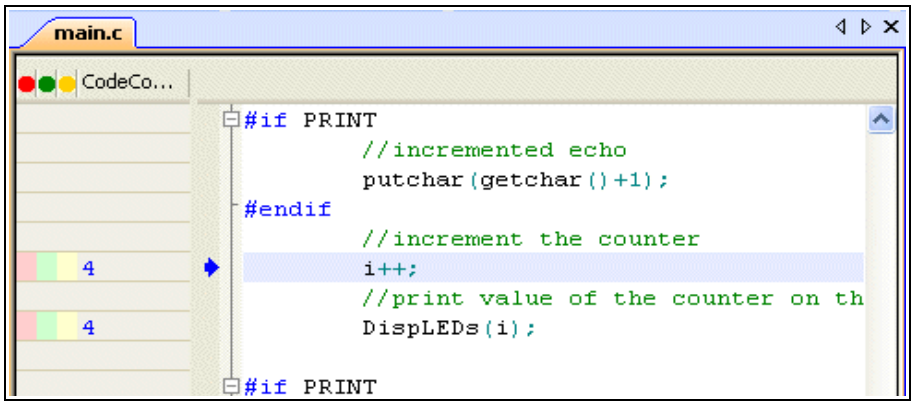


It shows the list of functions currently in the stack, allowing you to trace the calls up to the main function or the current **Interrupt Service Routine**. Double-click on a function in the stack view to place the cursor in the associated source file. There are a few restrictions for using this view:

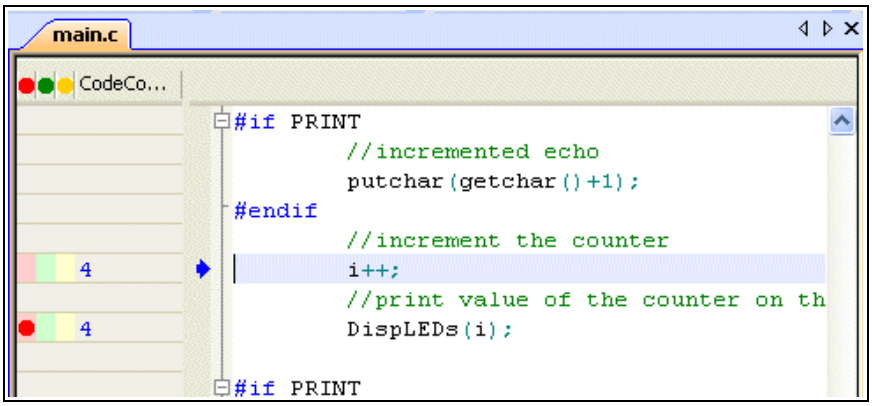
- It can only be used without optimization. (level zero).
- It needs debugging information for all the functions in the stack.
- It does not work during functions prologue and epilogue. (i.e. the very beginning and end of functions).
- It does not work properly when the current PC is not on the beginning of a C line. (after a stop or assembler step).

5.7 Using breakpoints

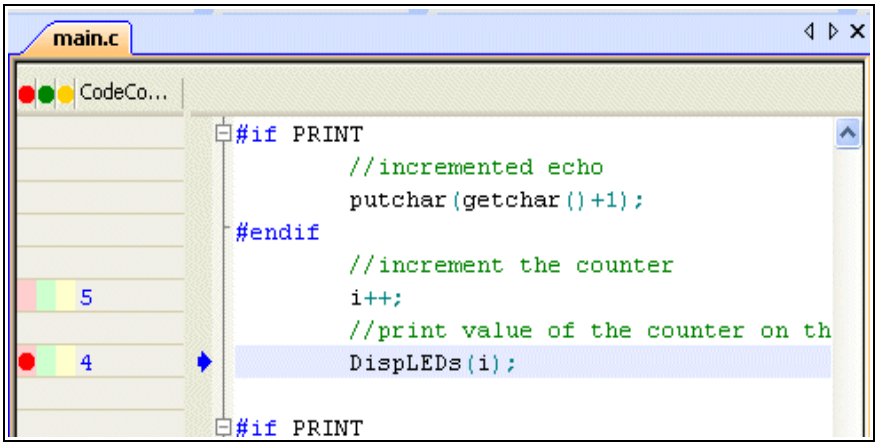
You can set a breakpoint either in the source file, or in the code view. Make sure that the application is not running (click on **PAUSE**). You can see on the left side of the source code, some lines with colored boxes (red green and yellow). These are code lines where a breakpoint can be set.



Then click on the red box **Toggle Breakpoint** and a red point appears on this line, which means that a breakpoint has been set on this line:



Then, click on **RUN** button and the application will stop running when this line is reached:



Refer to Ride7 general documentation for more information about the simulator user interface.

6. Debugging with hardware tools

In addition to the Raisonance simulator, Ride7 for ARM can be used with a number of hardware debugging tools.

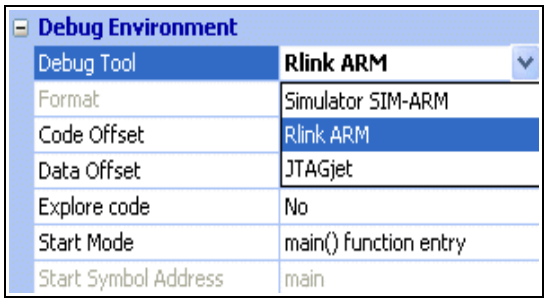
- **RLink** USB to JTAG emulators from Raisonance.
- **JTAGjet** USB to JTAG emulators from Signum Systems.
- **Serial Wire Viewer** debugging features for Cortex ARM based micro controllers. The SWV provides a low cost method of obtaining information from inside the MCU using ARM CoreSight™ technology.

From a user interface point of view, basic debugging functions (setting a breakpoint, single-stepping and checking memory and registers) are identical whether you are using the simulator or a hardware debugging tool. This chapter describes how to use the available drivers and the specificities of each.

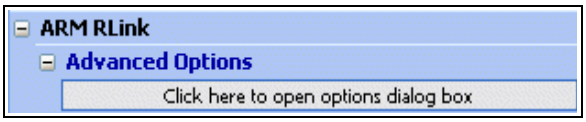
6.1 Selecting hardware debugging tools

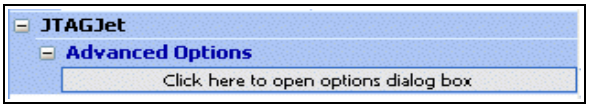
Within Ride7, you can choose your target hardware debugger from 2 menus.

- **Project Options** window.
- **Options > Project Properties** menu, **Advanced ARM Options > Debug Environment**.



1. From the **Debug Tool** option you can choose between a list of available tools.
   a. **RLink ARM**: If you have an RLink connected to the target ARM CPU on your application board via a JTAG or SWD connector, or if you are using the REva evaluation board, which includes an embedded RLink.
   b. **JTAGjet**: If you have a JTAGjet connected to the target STRx on your application board via either a JTAG or an ETM connector.
2. Then, configure the tool using the **Advanced Options**, either under the **ARM RLink** section or under the **JTAGJet** section:





6.2 RLink-ARM programming and debugging features

RLink is a USB to JTAG interface device designed by Raisonance. It allows programming and debugging of various microcontrollers, including all the ARM microcontrollers supported by Ride7 (see the up-to-date list in the **Advanced ARM Options > Processor > Processor name**).

RLink uses:

- JTAG protocol with ARM microcontrollers via the standard 20-point connector defined by ARM.
- SWD protocol with Cortex-M3 devices.

Before using RLink, make sure that you have installed the associated USB driver. Unless you have specified otherwise, it is installed along with Ride7. If the USB driver has not been installed, launch the program *RLinkUSBInstall.exe* (normally located here:*<Ride>\driver\RlinkDrv\RLinkUSBInstall.exe*

After running this program, Windows recognizes automatically when you plug an RLink in, the recognition could take some time on the first connection, but following connections of the same RLink on the same PC will be faster.

The REva evaluation board includes an embedded RLink. The whole board can be powered by the USB through the RLink. The target microcontrollers are on interchangeable daughter boards so that one evaluation board supports several different targets. For Ride7, there is no difference between operating the REva and using an RLink with any other board with the JTAG connector. See the REva documentation for more information.

6.2.1 RLink capabilities

RLinks have different capabilities for programming and debugging of ARM, ST7 and uPSD microcontrollers. Your RLink will fall into one of the following categories:

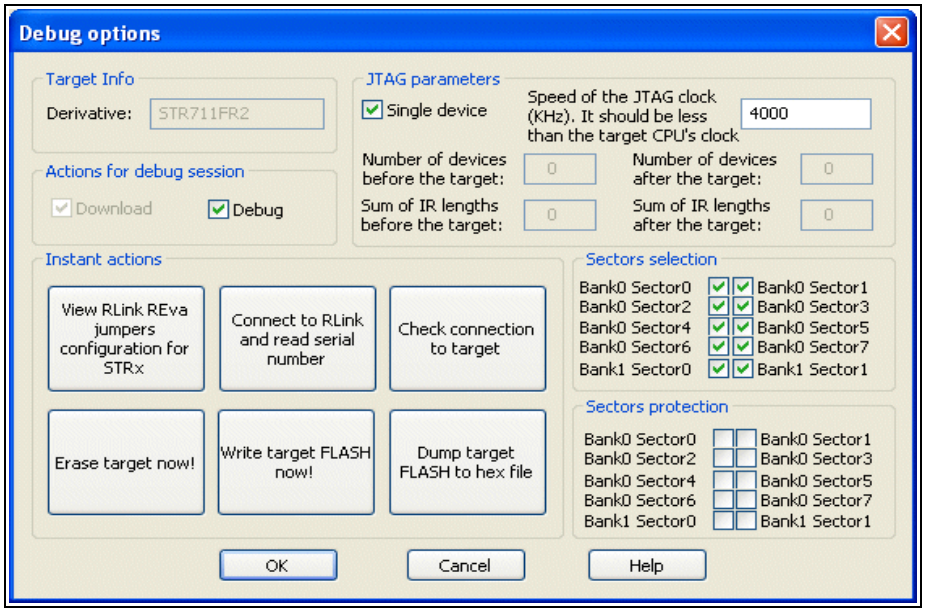
- Standard RLinks and RLinks in REva starter kits: Are allowed a limited access to ARM microcontrollers. With these RLinks, you can load and debug up to 64 Kbytes. You can also program (and execute) the full Flash memory, but you cannot debug it. They can also be used with all the other target CPUs supported by RLink (ST7, STM8, uPSD). Standard RLinks are in a grey plastic box. Starter kit RLinks are embedded in the REva evaluation boards contained in the REva starter kits. See the REva documentation for more information.
- Pro RLinks: Permit full access to ARM targets without any limitations. They can also be used with all the other ST targets supported by RLink (ST7, STM8, uPSD) without any limitations. They are in a plastic box for protection.

Your RLink's capability to program and debug any Ride7-supported target microcontroller can be reported when Ride7 reads your RLink's serial number. If you want to verify what kind of RLink you have, use the **Connect to RLink** instant action in the debug options (see next sub-section).

Note: RLinks and JTAGjet cannot work in external memory mode.

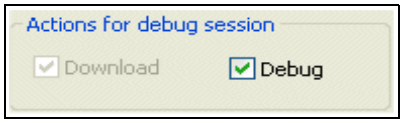
6.2.2 Configuring Ride7 for using the RLink

After selecting RLink-ARM as your debugging tool (see the section, *Selecting hardware debugging tools*), click on the **Advanced Options** button to open the **Debug options**.



Uncheck the **Debug** option if you want to use RLink as a simple programmer, e.g. if you want to run the application on the ARM without debugging it. Launching a debug session simply programs the code to your ARM CPU and starts execution, useful when using RLink without application source code.

To debug your application, make sure that **Debug** is checked as shown below.

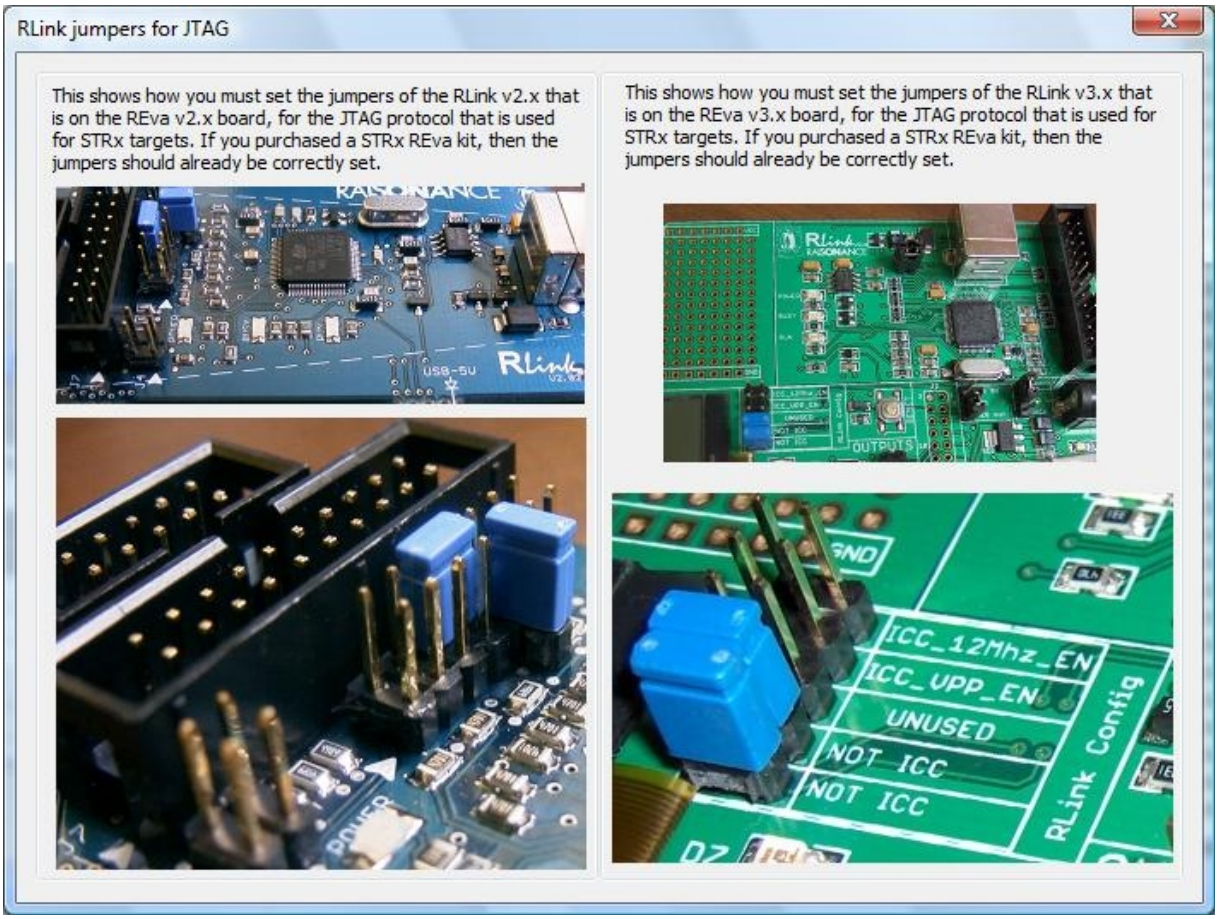


6.2.2.1 RLink-REva jumpers

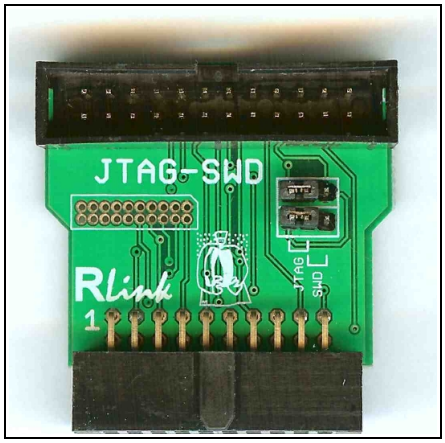
When using a starter kit's embedded RLink, ensure that your jumpers are set correctly on the RLink. To do this, click on **View RLink jumper configuration for ARM**.

If you purchased RLink as part of an ARM starter kit (such as the REva evaluation board for STR7), then the jumpers should already be correctly set. For this reason, you should only need to adjust these jumpers if they were accidentally unplugged, or if you are using an RLink that was configured for another microcontroller such as ST7.

These illustrations show the STRx configuration for the RLink jumpers. If these pictures differ from those in Ride7, assume that those shown in Ride7 documentation are correct.



6.2.2.2 RLink JTAG-SWD ADP jumpers

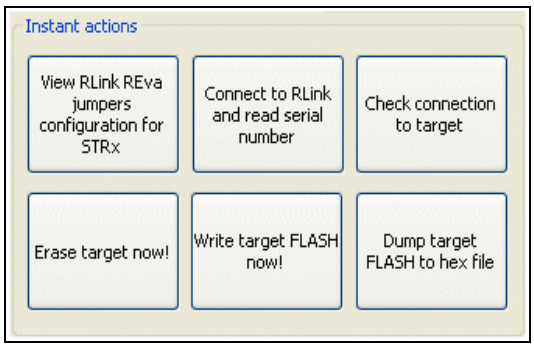


When using a stand-alone RLink, you must use an ADP to adapt the RLink's 24-pins connector to the target connector. If using JTAG-ARM ADP, there is no configuration to perform. If using JTAG-SWD ADP, you must plug two jumpers on the left side, as shown in the picture (default factory setting).

Note: SWD is supported by this version of the software. There is no need to change the jumpers position, keep them in the positions shown in this picture. (JTAG) As the SWD connector is a subset of the JTAG connector, you can also use the SWD protocol with the older JTAG-ARM ADPs, and also with all versions of the REva board.

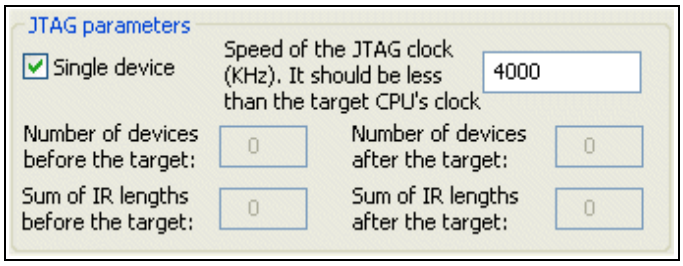
6.2.2.3 Instant actions

This allows you to carry out various instant actions without leaving this dialog box - useful for testing connections and retrieving information from the RLink and your ARM, and for programming the ARM and its configuration.



- **View RLink REva jumpers configuration for ARM** shows how you must set the jumpers on RLink for using it for programming and debugging ARM. You must be sure that the jumpers are correctly set before launching a debug session, or using any of the instant actions below.
- **Connect to RLink and read serial number** is useful for checking that RLink is working and properly connected and that the USB driver is correctly installed. It also allows you to read the RLink serial number, which you will be asked for if you contact our support team. When Ride7 checks the RLink serial number, the resulting serial number message includes information about your RLink's capabilities and limitations for the currently selected target microcontroller.
- **Check connection to target** allows you to read the JTAG IdCode of the chip. Use this to test the connections and power of the target ARM CPU.
- **Erase target now!** allows you to completely erase the target CPU's Flash (writing 0xFF).
- **Write target FLASH now!** programs the Flash with the current application's hex file generated by the linker. Then, launches the execution.
- **Dump target FLASH to hex file** reads the contents of the Flash and writes it in a file in hex format whose name is derived from the current application's name with the extension .hex (<*application name*>.*hex*).

6.2.2.4 JTAG parameters



- **Single Device**: specifies if there are several JTAG devices chained or only one device.
The JTAG standard makes it possible to chain JTAG devices (JTAG chaining is a complex process and should only be done if you have a good knowledge of the JTAG protocol). This section of the debugging options allows you to configure Ride7 to access an ARM microcontroller that is chained with other JTAG devices.
The checkbox should remain checked if there is no other device in the chain. Otherwise, you should uncheck it and enter the four parameters that the software needs to access the correct target in the chain. You need to know how many devices are in the chain, what order they are in, and the size of the IR register of each of them. Then, you must tell Ride7 how many devices are before (and after) the target in the chain. You must also tell it the sum of the lengths of the IR registers of the devices before and after the target.

- **Speed of the JTAG clock**: specifies the clock speed.
  If your CPU's clock is slow, then you must tell Ride7 to use a slow JTAG clock. If the JTAG clock is too fast compared with the target CPU's clock, then communication fails. This section of the debugging options allows you to specify the JTAG clock speed:
  Reducing the JTAG clock does not have very much influence on the programming and debugging speed because the USB is the bottleneck for most operations. Therefore, don't be afraid to use this option and enter the value of your target's clock speed in KHz.

Note: The RLink clock has a limited number of possible clock speeds. Ride7 selects the closest possible value that is less than the value you required. The minimum value is 400KHz. If your clock is slower than this, RLink might not be able to program and/or debug it. You must then purchase JTAGjet, or develop your application using a faster clock.

6.2.3 Hints and troubleshooting

6.2.3.1 Example projects

The examples in the REva folder of the **Ride7** directory are configured for use with the REva evaluation board, which includes the RLink. For standard installations they are found at *<Ride>\EXAMPLES\ARM\REva*. These examples can also be used with other demonstration and evaluation boards with a standard JTAG connector and the RLink. Before using an example, look at it and make sure that the jumpers on the REva evaluation board are set correctly (Enable switches for the LEDs, buttons, SCI, EEPROM, etc). Usually, there is some important information in comments at the beginning of the main file (I.e. the file that contains the "main" function).

The examples in the Primer folder of the Ride7 directory are configured for use with the Primer evaluation boards, such as the STM32-Primer, which includes the RLink. For standard installations they are found at:

<Ride>\EXAMPLES\ARM\Primer(\STM32).

These examples can also be used with other demonstration and evaluation boards with a standard JTAG connector and the RLink. Usually, there is some important information in comments at the beginning of the main file (I.e. the file that contains the "main" function).

6.2.3.2 Testing USB driver, connections and power supplies

Connect to RLink tests the USB driver installation and RLink operation. The RLink appears in Windows' device manager under the **Jungo** section when it is correctly recognized.

Check connection to target tests the connections and power of the target board and ARM CPU. This operation requires RLink to connect to the target ARM, ensuring that it is powered, correctly connected to RLink, and that the rest of the application board does not interfere with the communication between RLink and the ARM CPU (see below). It also checks that the target CPU is of the correct type.

6.3 JTAGjet programming and debugging features

Ride7 for ARM supports the JTAGjet emulator from Signum Systems Corp. Even though very similar in concept and usage, the JTAGjet is considerably faster than the RLink and offers a real-time trace (ETM) capability.

The user interface and advanced options for both emulators are the same and have the same functions.

JTAGjet is allowed full access to the STRx. This means that it can be used for programming and debugging, in Flash or in RAM, without limitation.

6.3.1 Signum Systems USB driver installation

The JTAGjet emulator requires installation of the Signum Systems USB driver. The installation process differs depending on whether you work in the Ride7 environment or use a third party debugger. In both instances, the driver is the same. The required driver files are on the CD-ROM disk that comes with the JTAGjet.

6.3.2 Installing the Signum Systems USB driver for Ride7

To install the USB driver for JTAGjet:

1. Insert the Signum Systems CD-ROM into your CD drive.
2. Plug the JTAGjet into the USB port.

Normally, MS Windows detects and installs the required driver automatically. If the OS cannot find the driver, direct it to the SigUSB.sys (SigUSB98.sys on Win98) file in the root folder of the Signum Systems CD-ROM. The latest USB driver version is also available from the Download page at www.signum.com.

For information on troubleshooting the USB driver installation process, refer to the “USB 2.0 Driver for JTAGjet and ADM51” installation instructions on the Signum Systems Technical Documentation web page at www.signum.com/tecdoc.htm.

6.3.3 Installing the RDI driver for third party debuggers

For instructions on installing the RDI driver for use with software tools other than those from Raisonance or Signum Systems, see “RDI Drivers for Third Party Debuggers with JTAGjet & RLink” installation instructions on the Signum Systems Technical Documentation web page at www.signum.com/tecdoc.htm. The procedure incorporates the installation of the USB driver.

6.3.4 Using the JTAGJet/STR9 trace features (ETM) in Ride7

To use the trace capability of the STR9 with JTAGJet, you should use the ETM connector instead of the 20-pin JTAG connector. See the ARM documentation for more information. You need a JTAGJet-trace (not all JTAGJets can perform ETM tracing) and a target board featuring the ETM connector.

The programming and debugging interfaces are the same as with the RLink or the standard JTAGJet, but, once the debug session has started, you will find additional commands in the menu **Debug > JTAGJet**.

If you are using the REva board with an ETM connector, you MUST remove the enable jumpers for LEDs D4, D5, D6 and D7, because they would interfere with the trace signals. These 4 LEDs cannot be used by the application while using the trace.

- The JTAGJet trace window: Selecting option **Open Trace Window** enables trace and opens the associated window. Refer to the Signum's Chameleon documentation for more information.
- The trace I/O configuration: The trace uses more pins than the simple JTAG debugging. See the ARM documentation for more information. These pins must be configured correctly for trace to work, and the user (debugged) application must not change this configuration.
  When using a JTAGJet with trace capability and an STR9 device, Ride7 configures the I/Os for the trace as described in a script file. Then, whenever the trace is enabled, and at every execution of some user code while the trace is enabled, it checks if the configuration has not been broken by the application. If it has, Ride7 notifies you and ask you if you want it to restore the configuration for trace, or if you want to disable the trace. If you choose to restore the trace settings, then Ride7 reconfigures the I/Os according to the data in the script file. This could affect your application’s behavior.
  The script is located in *<Ride installation directory>\bin\STR9_ETM_IO_conf.txt*. It is designed for the STR9 REva daughterboard, but can be modified for other boards. It consists of a list of entries containing Address, Value, InitMask and CheckMask. At reset, Ride7 writes the Value of each entry at the associated Address, masked using the associated InitMask. When enabling the trace or after some user code execution, Ride7 checks the value in the target device at each entry’s Address against the associated Value, masked using the associated CheckMask. Masks must be used to allow the user application to use some pins other than those used by the trace but on the same I/O port.

Note: Be **very careful** when modifying the script. A wrong script prevents you from using the trace, and could prevent you from debugging at all. Always make a copy of the original script before modifying it. Make sure you have all the necessary information about ETM from the ARM documentation to avoid any wrong manipulation.

6.4 Cortex Serial Wire Viewer (SWV) debugging features (Open4 RLink only)

Ride7 for ARM supports the SWV debugger for Cortex ARM based microcontrollers. The SWV uses the low cost ARM CoreSight™ technology to obtain information from inside an MCU.

6.4.1 Introduction

SWV is the ability of the ARM™ core to send out real-time trace information via a single wire port called the Serial Wire Output (SWO). This information is in several familiar formats (described in ARM documentation) such as:

- Instrumentation Trace Macrocell (ITM) for application driven trace source that supports printf style debugging.
  We call these traces ***Software traces***.
- Data Watchpoint and Trace (DWT) for variable monitoring and PC-sampling, which can in turn be used to periodically output the PC (sampled) or various CPU internal counters and to obtain profiling information from the target:
  - Program Counter sampling,
  - Data read and write cycles,
  - Variable and peripheral values,
  - Event counters,
  - Exception entry and return.

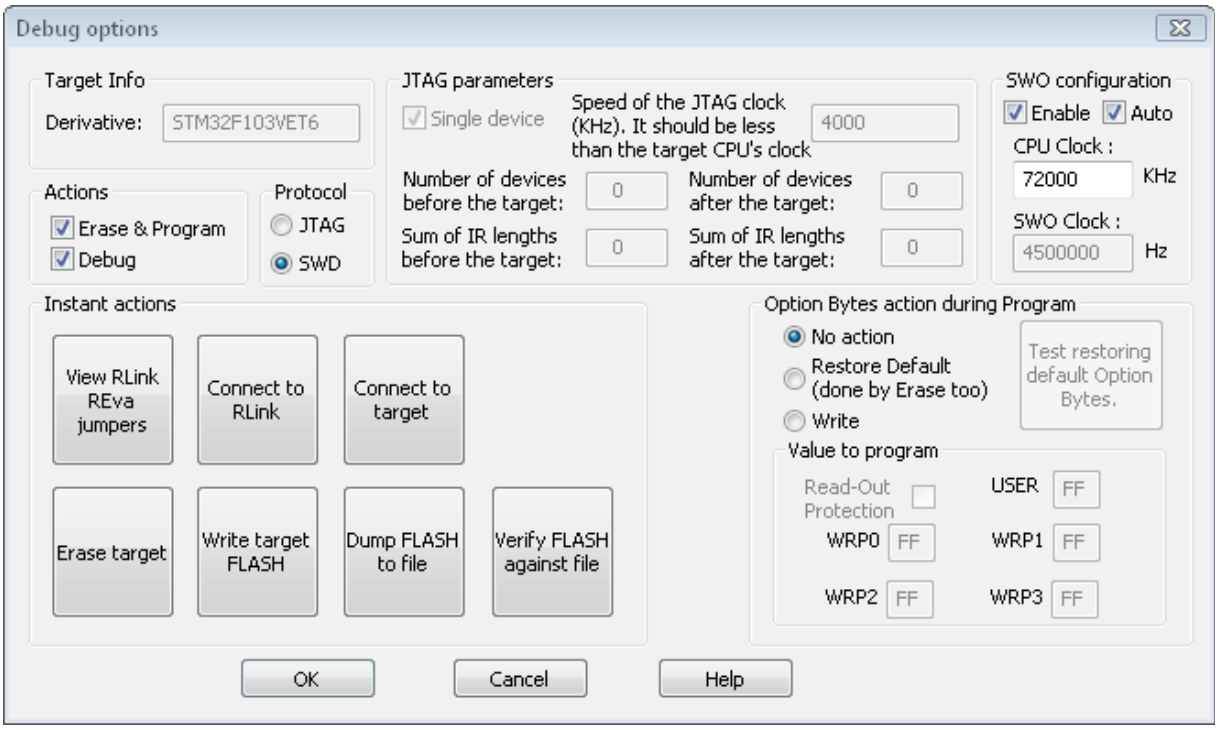
  We call these traces ***Hardware traces***.
- Timestamps and CPU cycles which are emitted relative to packets.

6.4.2 Hardware requirements

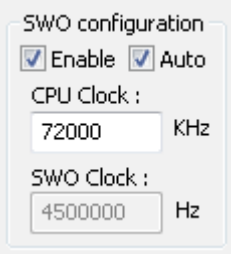
SWV is only available with a special RLink version of the debug probe, embedded for example, into the Evo and Open4 STM32 primers.

6.4.3 Configure Ride7 to use the SWV

Select **SWV** as debugging tool, and click on **Advanced Options** to open the **Debug options**.



This panel configures the Serial Wire Output:



- Check the **Enable** option if you want to use the SWO for SWV traces.
- Enter the CPU clock frequency of your target in kHz (72 MHz in this example). This information is necessary for the RLink to set the speed of the SWO asynchronous communication port.
- If you leave **Auto** checked, the system sets the maximum speed to 4.50 MHz (default value of SWO Clock). To change it, uncheck **Auto**, and enter the desired value in the **SWO Clock** field. SWO clock **must** be a sub-multiple of CPU clock, so this value may be adjusted by the system.

Note: Be careful that the CPU clock set in the configuration dialog box equals the real CPU clock of your target, or else the data will not be correctly interpreted.

6.4.4 Modify your application to use SWV software traces

Before sending data to the debugger through the SWO, you must add some code to your application. Use the **`ITM_SendChar`** function (provided by the CMSIS library for printf-style output) to send a character string to the ITM channel n° 0.

Example 1:

```
while(*str)
    {
    ITM_SendChar(*str);
    str++;
    }
```

Example 2:

You can also send data of different types (byte, half word or word) to the channel of your choice (0 to 31) using the function as described below:

```
// Write to the SWO through a specific ITM port
WriteITM(uint32_t val, uint32_t port)
    {
       if ( (CoreDebug->DEMCR & CoreDebug_DEMCR_TRCENA)     // Trace enable ?
       &&   (ITM->TCR & ITM_TCR_ITMENA)                     // ITM enable ?
       &&   (ITM->TER  & (1UL << port)) )                   // ITM port enable ?
         {
       while (ITM->PORT[port].u32 == 0);                    // ITM port free ?
//     ITM->PORT[port].u32 = val;                           // Write 32 bits value
//     ITM->PORT[port].u16 = (u16) val;                     // Write 16 bits value
       ITM->PORT[port].u8 = (u8) val;                       // Write 8 bits value
         }
    }
```

This function verifies if the trace, ITM modules and the desired channel are enabled. It then waits for the channel release and sends 8-, 16- or 32-bit values.

Note: Two ITM communication channels are used by CMSIS to output the following information:
- ITM channel 0: for printf-style output via the debug interface,
- ITM channel 31: reserved for RTOS kernel awareness debugging.

Exampe 3:

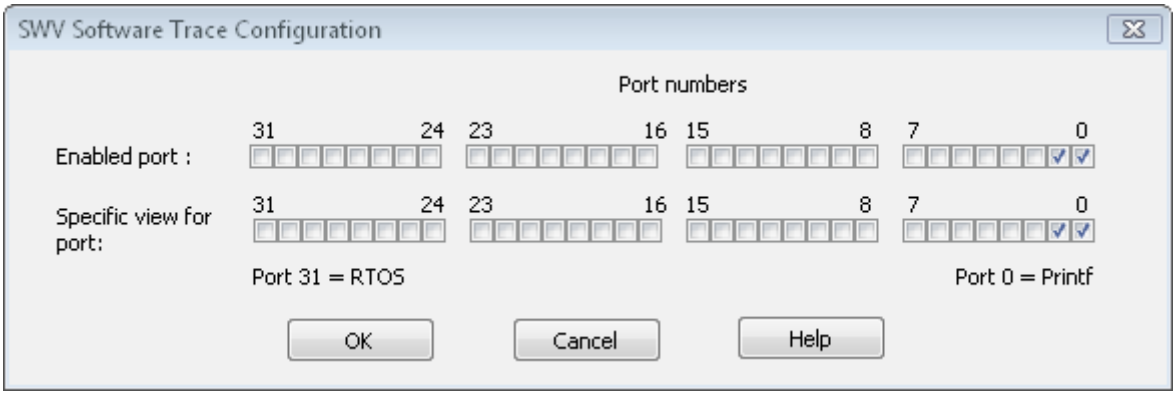
Look at the **TestSWV** application in {*RideDir*}*Examples\ARM\Primer\STM32EvoPrimer\Test_SWV*

This application periodically sends the message “Hello world !”, and the value of the loop counter. It implements a **`print`** function that prints strings with integers inside (%d formatter), and sends this string with the **`ITM_SendChar`** function. It also uses the previously described **`WriteITM`** function.

6.4.5 Configure Ride7 to use SWV software traces

After beginning your debug session, select the ITM channels to use and the associated views.

Go to the menu **Debug**->**Advanced commands...**->**SWV Software traces configuration** to open the **SWV Software Trace Configuration** dialog box:



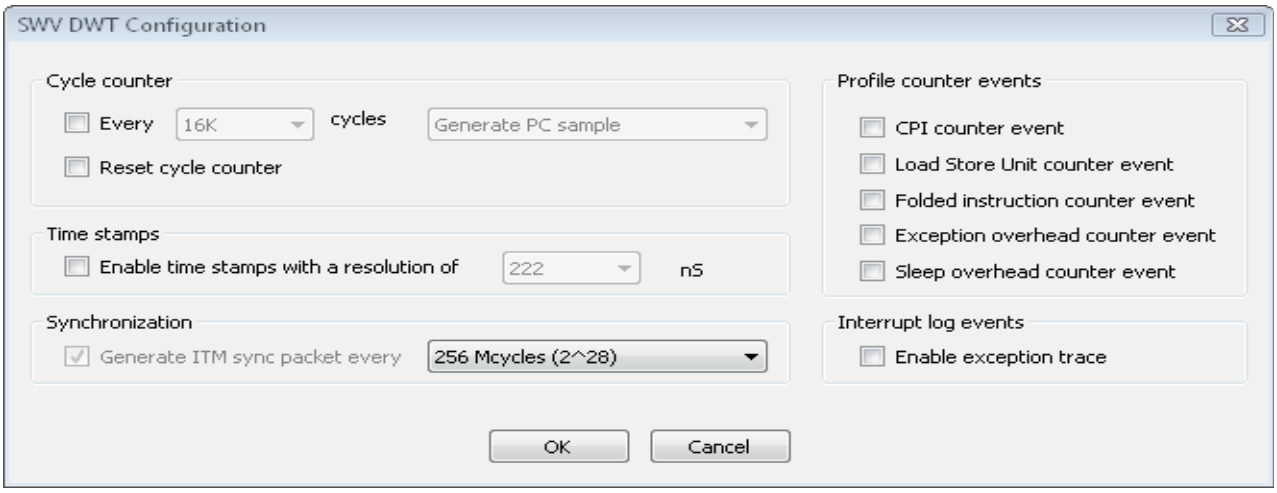
You can enable each ITM channel by checking the corresponding **Enabled port** check box. If they are not checked, the application program will not be able to send data through this port.

You can open a specific view for this channel by checking the corresponding **Specific view** check box. If you do not, the trace for this channel will be only available on the **Global Trace View**.

Note: These configurations are saved within the project.

6.4.6 Configuring Ride7 to use the SWV hardware traces

Go to the menu **Debug**->**Advanced commands...**->**SWV Hardware traces configuration** to open the **SWV DWT Configuration** dialog box:



Cycle counter:

- Every x cycles: enables periodic sampling of the Program Counter or the Cycle Counter value, based on the CPU Cycle Counter. The period of the event can be choose from 64 cycles up to 16K cycles. The highest values will probably not work because the SWO communication bandwidth is not big enough to handle that much flow of data.
- Reset cycle counter: causes the cycle counter to be set to zero, when validating the dialog box by clicking on the OK button.

Time stamps: the Cortex debug module enables time stamp for every ITM or DWT event or group of events. Use the resolution drop-down list to choose the resolution of the time stamp value. Four values are available: CPU clock / 1; CPU clock / 4; CPU clock / 16; CPU clock / 64.
These correspond, for example, to 13ns, 55ns, 222ns, 888ns for a 72MHz CPU clock.

The lowest resolution is only useful if the time between each event packet is long: a time stamp is automatically sent if there is no event and the time counter overflows (every 26ms, 110ms, 440ms or 1776ms at 72MHz).

Synchronization: causes ITM synchronization packet to be sent every time the specified period elapses. It allows the system to resynchronize with the SWV flow in case of lost events, or bad event reception. Three periods are available: 16M; 64M; 256M cycles.
These correspond, for example, to 233ms, 933ms and 3.7s at 72MHz.

To stop losing events, you can increase the synchronization period, but be aware that this also increases the time before resynchronizations.

Profile counter events: the DWT unit contains a few profiling counters which cause trace samples to be generated on overflows of these counters (every 256 of corresponding instructions).

Five counters can be monitored individually:

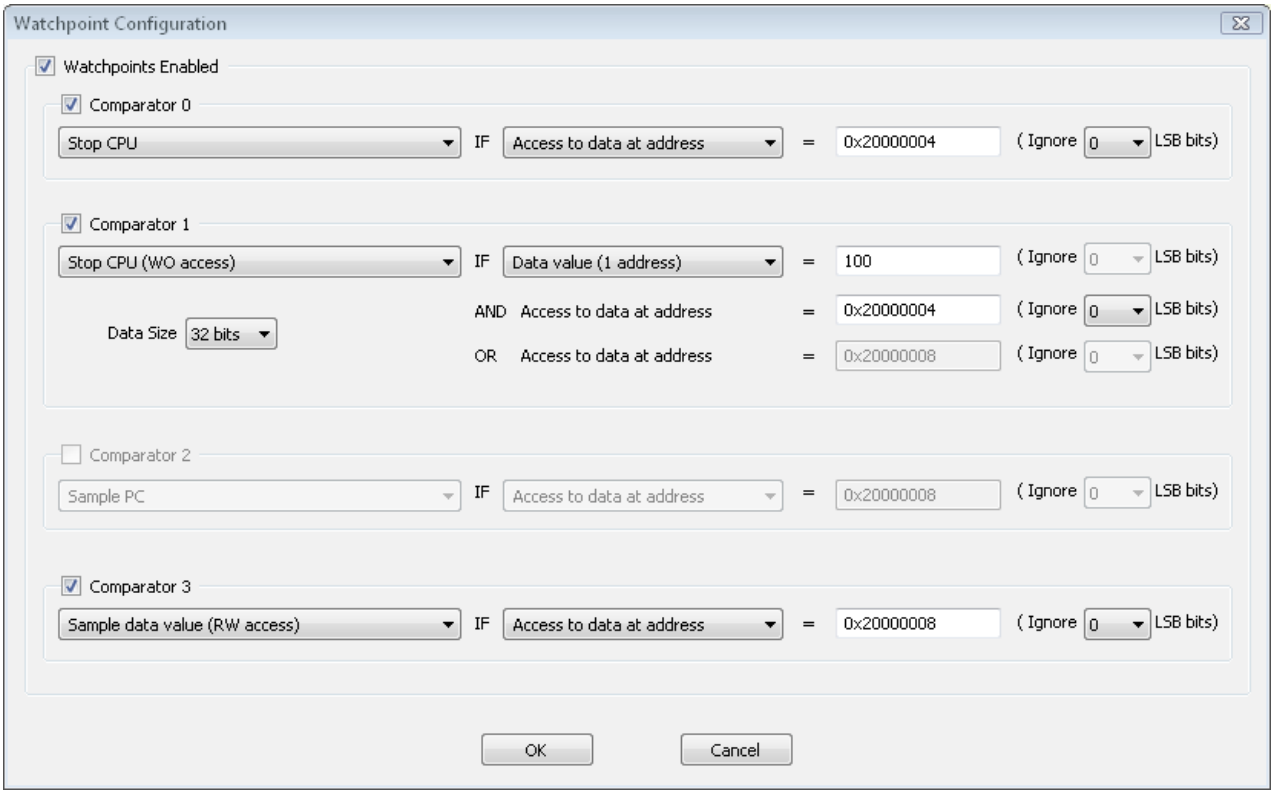
- Folded instructions counter,
- Load Store Unit operation counter,
- Sleep overhead counter,
- Interrupt overhead counter,
- CPI counter.

Interrupt log events: turn on tracing of exception entries and exits.

Note: These configurations are saved within the project.

6.4.7 Configuring Ride7 to use the SWV watchpoint traces

Go to the menu **Debug->Advanced commands...->SWV watchpoints configuration** to open the **Watchpoint Configuration**: This window configures watchpoints and data watch via four comparators.



Watchpoints Enabled: This is a global enable/disable control for all comparators. If this is not checked then all comparators are disabled.

Comparator X: Each comparator can be individually enabled and configured ,and generate an action when a comparison match occurs.

Match condition:

A drop-down box selects the match condition between several choices:

- *Cycle counter*: generates a trace event when the Cycle Counter reaches the parametrized value (**Comparator 0 only**).
- *Program counter*: generates a trace event when the Program Counter reaches the parametrized value (equivalent to a code breakpoint, if Stop CPU is configured as action).
- *Access to data at address*: generates a trace event when the CPU accesses the memory address equal to the parametrized value.
- *Data value (1 address)*: generates a trace event when the CPU accesses the memory address equal to the second parametrized value, **and** that data value is equal to the first parameter value (**Comparator 1 only**).
- *Data value (2 addresses)*: generates a trace event when the CPU accesses the memory address equal to the second or third parameter value, and that data value is equal to the first parameter value.(**Comparator 1 only**).

Note for Data Value matching (Comparator 1):

- When Data Value matching 1 address is used, Comparator 2 is implicitly used, and is not available for another match condition.

- When Data Value matching 2 addresses is used, the Comparator 2 and 3 are implicitly used, and are no longer available for another match condition. These two addresses are not connected and do not form any range together.

Action: A drop-down box allows selection of the action:

- *Sample PC*: generates a trace event containing the address of the instruction that was executing at the moment of the matching condition, during read or write access.
- *Sample data address (RW access)*: generates a trace event containing the lower 16 bits of the address of the memory location to which data access was being performed by the CPU at the moment of the matching condition, during read or write access.
- *Sample data value (RW access)*: generates a trace event containing the data value of the memory location to which data access was being performed by the CPU at the moment of the matching condition, during read or write access.
- *Sample data address + data value (RW access)*: generates two trace events, as defined above (data address and data value).
- *Sample PC + data value*: generates two trace events, as defined above (Sample PC and data value).
- *Stop CPU*: generate a watchpoint at the moment of the matching condition, during Read or Write access.
- *Stop CPU (RO access)*: generate a watchpoint at the moment of the matching condition, only during read access.
- *Stop CPU (WO access)*: generate a watchpoint at the moment of the matching condition, only during write access.
- *Generate ETM event trigger*: generates a ETM trigger at the moment of the matching condition, during read or write access.
- *Generate ETM event trigger (RO access)*: generates a ETM trigger at the moment of the matching condition, only during read access.
- *Generate ETM event trigger (WO access)*: generates a ETM trigger at the moment of the matching condition, only during write access.

Ignore: A drop-down box allows selection of the bit length of the mask applied to the address: it specifies the number of least significant address bits to be ignored by the comparator during address match, from 0 to 15, in order to form a range value condition.

Data size: A drop-down box selects the data size for data value matching: 8 bits, 16 bits or 32 bits.

Notes:

The available functionalities depend on the hardware implementation of the connected target CPU.

When trace is enabled, watchpoints must be configured by this dialog box, they are no longer available on **Memory View**.

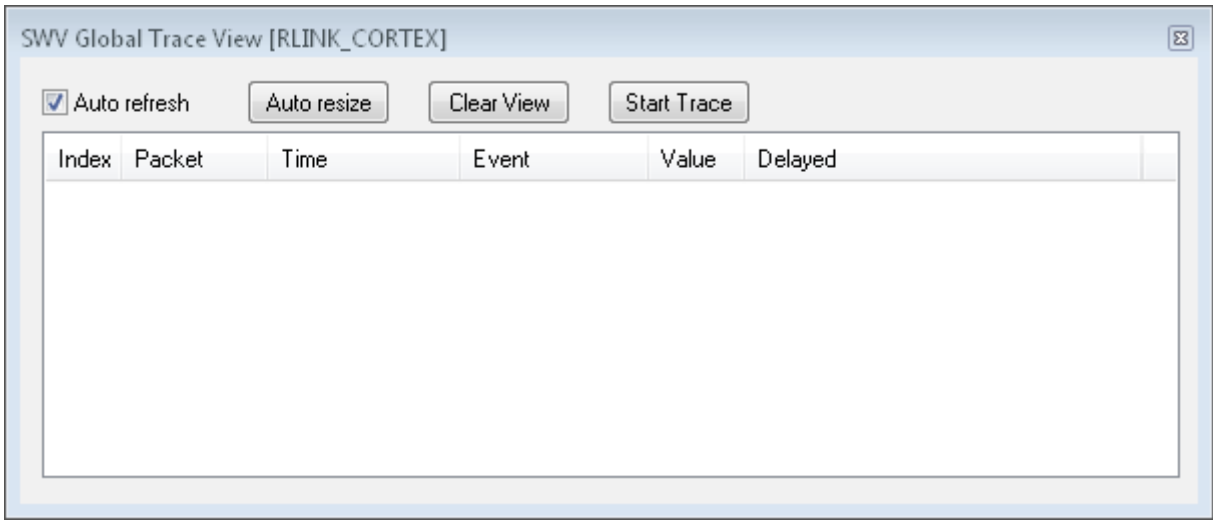
All these configurations are saved within the project, except the global enable: the watchpoints are systematically disabled at the beginning of the debug session.

The ETM event triggers are only for further use as control inputs to ETM trace hardware.

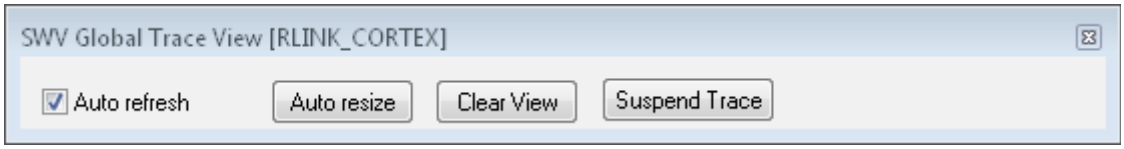
6.4.8 Start / Stop the trace

The Start / Stop command is available on the **SWV Global Trace View**.

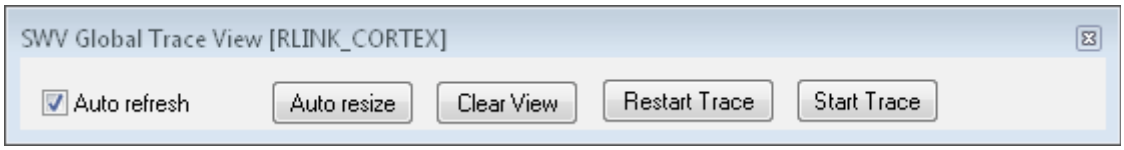
1. Click on **Debug->Advanced commands...->SWV Global Trace View** .



2. Click on **Start Trace** to start the trace acquisition.



3. Click on **Suspend Trace** to stop the trace acquisition.



4. Click on **Start Trace** again to continue trace acquisition: the new events are appended after the existing trace.
5. Click on **Restart Trace** to start a new trace acquisition: the trace is reinitialized (views and buffers are cleared) and the previous trace is lost.

6.4.9 Visualizing SWV traces with Ride7

Several views visualize traces during debug, whatever the CPU status, running or stopped.

6.4.9.1 SWV Software Trace View

Menu **Debug**->**Advanced commands...**->**SWV Software Trace View** opens the **SWV Software Trace View** pane. This contains all the views of the channels that you configured with specific views, one per tab.

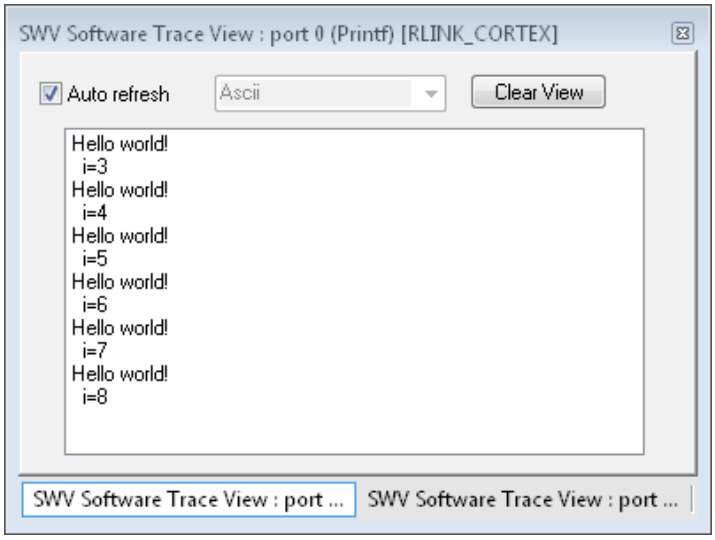
Available commands:

- **Auto refresh**: uncheck this option if you want to temporarily navigate into the view.
- **Display choice**: you can change the display mode via a combo box.
  Available modes are decimal or hexadecimal.
- **Clear View**: clears the view, without stop the trace.

Here an example of port 0 is shown in Ascii format.

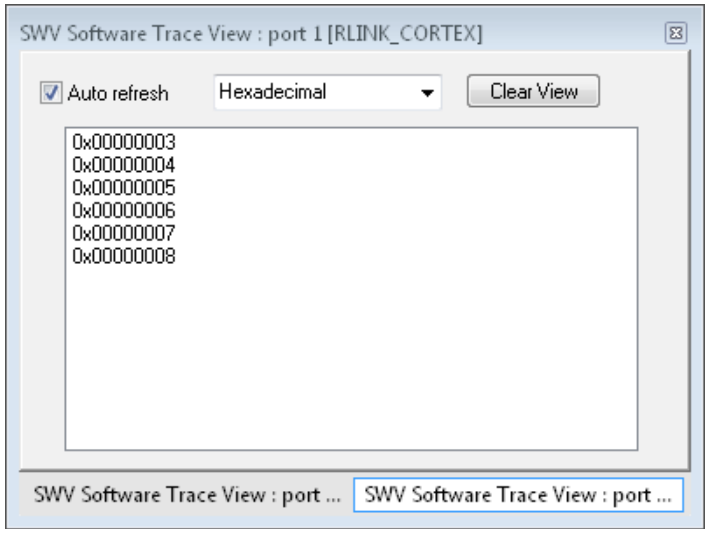
Channel 0 is reserved for printf-type output, so this view only displays Ascii information (the combo box is disabled).

Channel 0 expects a line feed character (\n = 0x0A) before displaying an entire line.



Here an example of port 1 is shown in Hex format.

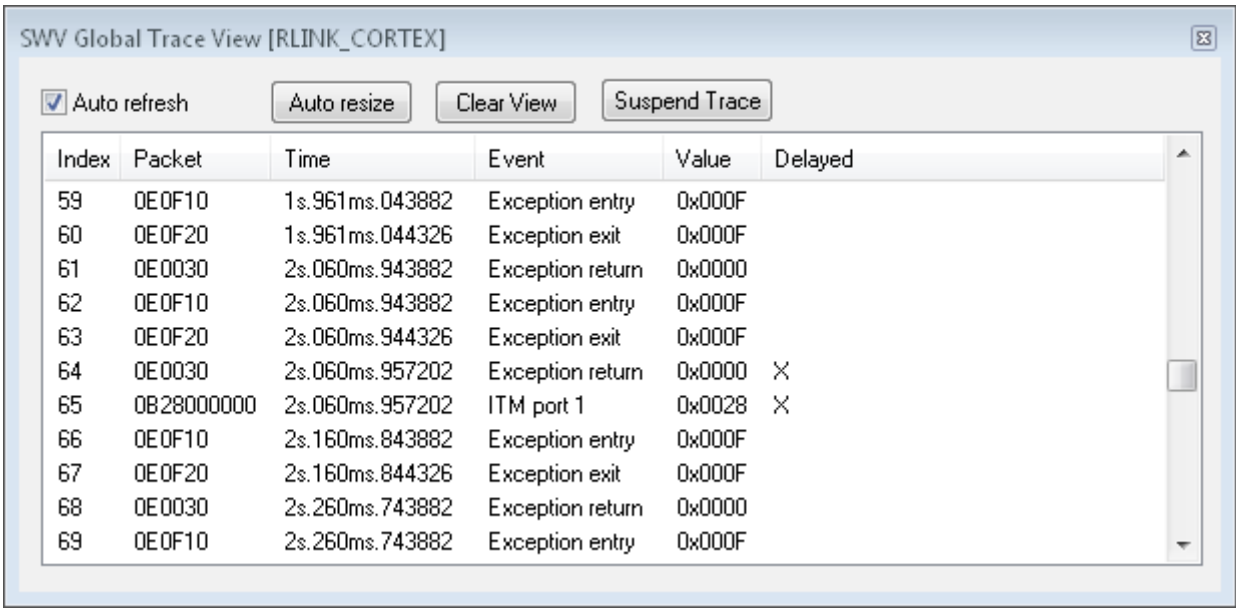
These other ports can be displayed in Hexadecimal, Decimal or Ascii format



Note: If you have already opened a **Software Trace View** and you add a new channel (using the **Enable** and **Specific View** in the **SWV Software Trace configuration** dialog box), you must reload the **Debug->Advanced commands...->SWV Software Trace View** item menu to open the new view.

6.4.9.2 SWV Global Trace View

Debug->Advanced commands...->SWV Global Trace View opens the **SWV Global Trace View**:



This view is global for all ITM channels and DWT features. It shows all the events received on the SWV, one event by line, with the following information:

- **Index**: incremental event number,
- **Packet**: packet data as it was received, with no treatment,
- **Time**: time stamp of the event,
- **Event**: type of event with channel source number,
- **Value**: value sent by the event (hex value for ITM Port 1, and exception number in this example),
- **Delayed**: indicates that the time is not accurate, the SWV module has not been able to set a time to all the events that occurred (the Time Stamp event has lower priority than other events).

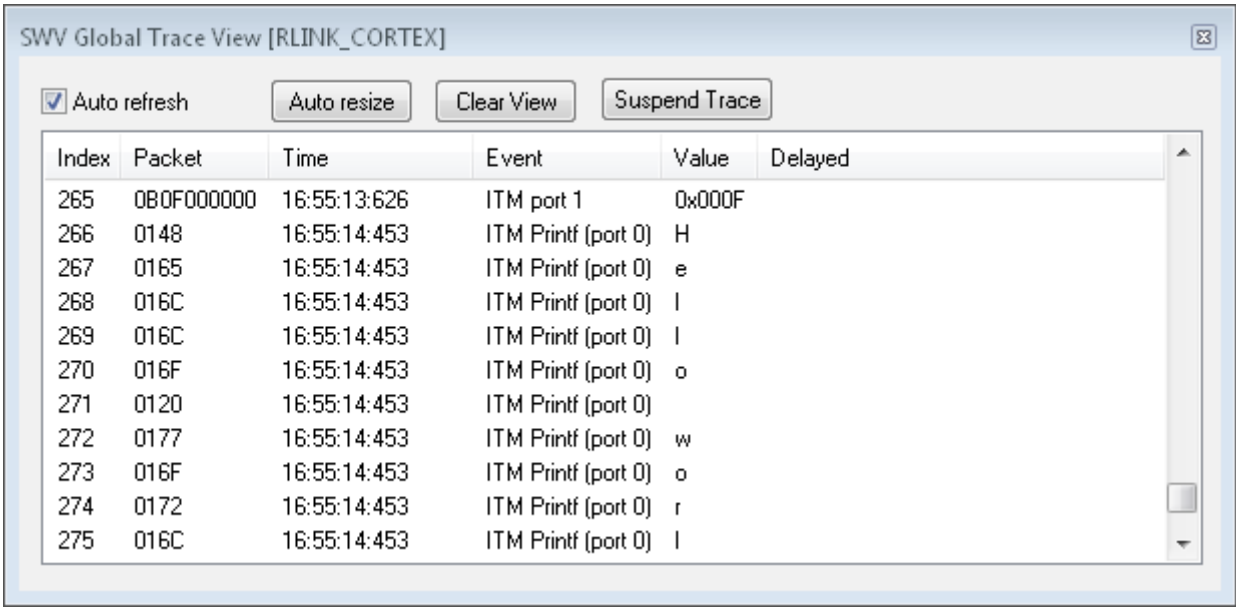
Available commands:

- **Auto refresh**: uncheck this option to temporarily navigate into the view.
- **Auto resize**: push this button to resize all the columns of the list, and to display all the data.
- **Clear view**: allows to clear the view, without stop the trace.
- **Suspend Trace / Restart Trace:** stops the trace, or reinitializes the trace.
- **Start trace**: start the trace after a suspend command.

Notes:

- Time is relative to the beginning of the trace, if Time Stamp has been validated in the DWT configuration dialog box. In other cases, the time is absolute PC time.
- When you click on **Start Trace**, after having stopped the trace, trace acquisition is restarted, the new trace events are appended at the end of the previous trace.
- When you click on **Restart Trace** after having stopped the trace, the trace is reinitialized (views and buffers are cleared) and the previous trace is lost.

Example of trace with PC time stamp:



6.4.9.3 SWV error messages

Some error messages can occur in the **Debug Output View**, during an SWV session:

- ***SWV unknown trace received (check CPU frequency or reduce trace speed)***: the system received data that it could not interpret. It may be a configuration or synchronization problem. The SWV data flow depends on the CPU clock, so check that the CPU clock indicated in the configuration dialog box equals the real CPU clock of your target.
  If the frequency is correct, try to reduce the trace speed.
  If the problem persists, stop the application for a while, and run it again.
  If the problem still persists, end the debug session and restart it.
- ***SWV overrun:*** Ride7 is not able to treat all the data flow, and some data is lost. Reduce the frequency of sending or quantity of data sent by the application software.
- ***SWV buffer full***: the size of the acquisition buffer is limited to 100 KB. When it becomes full, this message appears in the Debug Output View, and acquisition is stopped.
- ***SWV USB communication error. Please check that the dongle is connected.***: some communication errors occurred. Check the RLink connection.

Some special events can occur in the global trace view:

- ***Internal Cortex fifo OVERFLOW***: the fifo of the CPU SWV module overruns. Reduce the frequency of sending or quantity of data sent by the application software.
- ***RLink internal buffer OVERRUN****:* Ride7 is not able to treat all the data flow, and some data is lost. Reduce the frequency of sending or quantity of data sent by the application software
- ***Unknown event:*** the system received data that it could not interpret. See the ***SWV unknown trace received*** above***.***

6.4.9.4 SWV Limitations

Note: Remember that SWV is a low cost method for CPU tracing.

Clearly a single serial wire port cannot provide all trace information because of the high speed of the Cortex M3 micro-controllers. In particular SWO and Ride7 are not capable of providing every program counter value because of the high speed at which the STM32 runs.
The system is also unable to provide all exception or diagnostic counters events, depending on the target CPU clock and the interrupt frequencies.

7. Raisonance solutions for ARM upgrades

	Standard Software Features		Advanced Software Features	
	Entry	Pro	Entry	Enterprise
Hardware Product	with RLink-STD	with RLink-PRO	with RLink-STD	with RLink-PRO
• Unlimited programming	☒	☒	☒	☒
• Unlimited debugging	☐[1]	☒	☐[1]	☒
• SWV Trace	☐[2]	☐[2]	☐[2]	☐[2]
Software Product	RKit-ARM-Lite	RKit-ARM-Lite	RKit-ARM-Enterprise	RKit-ARM-Enterprise
• Support multiple MCU manufacturers	☒	☒	☒	☒
• GCC Compiler	☒	☒	☒	☒
• Integrated GCC compiler control	☒	☒	☒	☒
• Integrated 3rd-party compiler control	☐	☐	☒	☒
• Ride7 debug controls (run, step,...)	☒	☒		☒
• Ride7 debug views (memory, stack, variables, ...)	☒	☒		☒
• Ride7 simulator	☒	☒		☒
• Ride7 standard editor	☒	☒		☒
• C++ debugging	☐	☐		☒
• Auto C formatter	☐	☐		☒
• Integrated calculator	☐	☐		☒
• Comment stripper	☐	☐		☒
• Unix line-end converter	☐	☐		☒
• Ride7 standard project manager	☒	☒		☒
• Script-based controls (C, C++, Jscript)	☐	☐	☒	☒
• Integrated CVS version managment	☐	☐	☒	☒
• RFlasher programming interface	☒	☒	☒	☒
• FAQ	☒	☒	☒	☒
• Software updates	☒	☒	☒	☒
• Forum	☒	☒	☒	☒
• Direct support (phone, email)	☐	☐	☒	☒
• 2-day Response guarantee	☐	☐	☒	☒

1a 1b 2

By default, all Raisonance hardware products are sold with the RKit-ARM-Lite software product included, unless you ordered the RKit-ARM-Enterprise software product.

You can upgrade your Raisonance tools for ARM in 2 separate ways:

- Upgrades 1a and 1b unlock the 64 Kbyte debug limitation.
- Upgrade 2 provides the advanced software features of the RKit-ARM-Enterprise software.

7.1 Hardware upgrade paths 1a and 1b (RLink-STD to RLink-PRO)

Debug code size limitations are specific to, and controlled by, the licenses of the RLink hardware. The RLink acts as a license dongle to facilitate portability of the unlimited debugging capability from one computer to another. For more information about licenses, refer to the RLink User Manual.

- Most Raisonance hardware products for ARM have the capabilities of the RLink-STD and are subject to a 64 Kbyte debugging code-size limitation for ARM.
- RLink-PRO, STM32 Primer1-PRO, STM32 Primer2-PRO and products for which users have bought an upgrade license (note 1) are not subject to this limitation.

Upgrades to unlock the debug limitation are available to users of the RLink-STD (including Open4, Primers and REva). This upgrade has no other effect or impact on the software features of the respective RKit.

If the hardware has been upgraded, you need to install an upgrade license file that is specific to your hardware's serial number in a directory on any computer that you will use with the Rlink-PRO. "PRO" licenses cannot be transferred to another RLink or hardware device. If you require an upgrade for another hardware product, please contact Raisonance sales or your Raisonance distributor.

7.2 Upgrade path 2 (RKit-ARM-Lite to RKit-ARM-Enterprise)

The RKit-ARM capabilities are determined by a software-based license and are independent of the hardware that is used with them. Software licenses are node-locked licenses - specific to a computer. However, a dongle-based option is available when ordering. As shown here, the debugging code size limitation is based on the Raisonance hardware product (RLink) and is not affected by the RKit-ARM license.

- The RKit-ARM-Enterprise software license allows access to the features listed in the illustration above.
- The RKit-ARM-Lite is provided by default with all Raisonance hardware products. An upgrade path (Arrow 2) is provided to enable you to access the features of the "Enterprise" software.

7.3 RKit and RLink options

RKit-ARM-Lite

- All supported ARM sub-families
- GCC toolset
- GUI interface for compiler control
- Project manager
- Debug run control, breakpoints and all views
- Full programming GUI
- Support via forums, email with standard priority

Rkit-ARM-Enterprise (All features of the “Lite”, plus...)

- Phyton compiler support
- Script-based controls (C, C++, JScript)
- Control of version management tools (CVS, ...)
- Automatic C formatter
- Calculator (standard & hex)
- Comment stripper
- Unix line converter
- Support via forums, email with high priority

RLink-STD

- Debugging – limited to up to 64 Kbytes (RAM / Flash)
- Programming – No code size limitation

RLink-PRO

- Debugging – No code size limitation
- Programming – No code size limitation

8. Conformity

ROHS Compliance

Raisonance products are certified to comply with the European Union RoHS (Restriction of Hazardous Substances) Directive (2002/95/EC) which restricts the use of six hazardous chemicals in its products for the protection of human health and the environment.

The restricted substances are as follows: lead, mercury, cadmium, hexavalent chromium, polybrominated biphenyls (PBB), and polybrominated diphenyl ethers (PBDE).

CE Compliance

Raisonance products are certified to comply with the European Union CE Directive.

In a domestic environment, the user is responsible for taking protective measures from possible radio interference the products may cause.

FCC Compliance

Raisonance products are certified as Class A products in compliance with the American FCC requirements.

In a domestic environment, the user is responsible for taking protective measures from possible radio interference the products may cause.

WEEE Compliance

Raisonance disposes of its electrical equipment according to the WEEE Directive (2002/96/EC). Upon request, Raisonance can recycle customer's redundant products.

For more information on conformity and recycling, please visit the Raisonance website at: http://www.raisonance.com

9. Glossary

Term	Description
SWD	Serial Wire Debugger
JTAG	Joint Test Action Group (or "IEEE Standard 1149.1"). A standard specifying how to control and monitor the pins of compliant devices on a printed circuit board.
Ride7	Raisonance integrated development environment
ETM	Embedded Trace Macrocell
ARM	Advanced RISC Machine
RLink	Raisonance's in-circuit debugging and programming tool
REva	Raisonance's universal evaluation board
STR7	STMicroelectronics 32-bit microcontroller family with ARM7 RISC 32-bit CPU core
STR9	STMicroelectronics 32-bit microcontroller family with ARM966E-S CPU core
STM32	STMicroelectronics 32-bit Flash microcontrollers with ARM Cortex™-M3 core
LPC	NXP microcontrollers, some of them including an ARM7 RISC 32-bit CPU core
SWV	Serial Wire Viewer

10. Index

Alphabetical Index

11. History

Date	Description
January 2009	Initial version
June 2010	Added SWV Increase debug limit up to 64Kb
October 2010	Corrections in the list of derivatives.
March 2011	Completed SWV with DWT and watchpoints. Corrected RAM boot mode information.
June 2011	Added generic compiler interface, new licensing/registering process.
September 2011	Documented the Phyton Compiler interface.